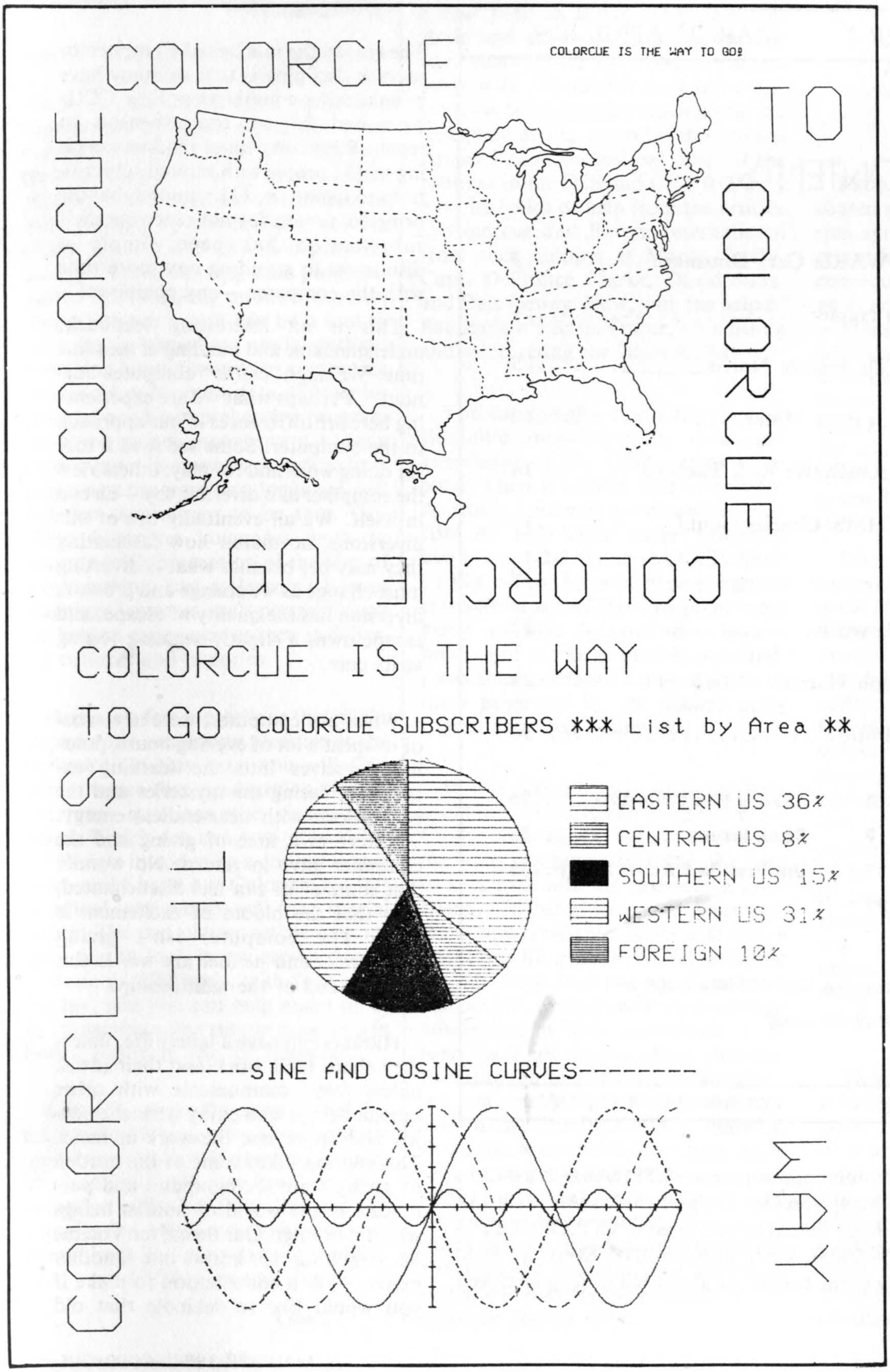
COLORCUE

VOLUME VI
NUMBER 2

A BI-MONTHLY PUBLICATION BY AND FOR INTECOLOR AND COMPUCOLOR USERS



tiny-**PASCAL**

EPSON MX80 GRAPHICS

ASCII, MASKS and BCD

USER SUPPORTED SOFTWARE

Colorcue

VOLUME VI, NUMBER 2 MARCH/APRIL 1984

CONTENTS

USER SUPPORTED SOFTWARE: Gary Dinsmore 3

A RETURN TO FCS: Tom Napier 4

ASCII, MASKS, AND BCD: Joseph Norris_____.....5

COMPILING BASIC, Part 3: Peter Hiner 10

COMPUCOLOR MEETS MORROW: Rick Taubold 14

STRUCTURED ALGORITHMS: Charles Gould 18

CYPHER: W.S. Whilly 19

EPSON GRAPHICS: Rice Lowe 22

PRODUCT REVIEWS: Joseph Norris 24

TINY-PASCAL: Doug VanPutte 30

Editor's Desk 2
Colorcue Contest 9
Suggestions for Articles...17
Refresher Course 21
Basic's File Structure 26
Back Issues 33
Unclassified Advertising ..35

COVER: Gleanings from the CGP-115 (see page 24.)

BACK: A useful chart by Ric Lowe

EDITOR: JOSEPH NORRIS COMPUSERVE: 71106, 1302

COLORCUE is published bi-monthly. Subscription rates are US$18/year in the U.S., Canada, and Mexico (via First Class mail), and US$30 elsewhere (via Air Mail). All editorial and subscription correspondence should be addressed to COLORCUE, 19 West Second Street, Moorestown, NJ 08057, USA. (609-234-8117) Every article in COLORCUE is checked for accuracy to the best of our ability but is not guaranteed to be error free.

"Carts don't belong before horses."

The reason the number of Compucolor users is declining is that so many have bought other computers or their CCIIs have had failures too expensive to repair. Right? In a quasi-random sampling of 154 people with expired subscriptions to Colorcue, 121 reported that the principle reason for not renewing their subscription has been simply a disinterest in spending any more time with the computer —any computer!

This is an interesting feedback, understandable and puzzling at the same time. We might call it "computer burnout". Perhaps what we are experiencing here are differences in our approach to the computer. Some see it as a tool for doing work and for play; others view the computer as a diverting toy —an end in itself. We all eventually tire of our diversions, no matter how fascinating they may be, because what is diverting must change as we change and grow. A diversion has the quality of escape, and escape owns a slightly negative feeling at its core.

When our computers were new most of us spent a lot of evening hours, pouring ourselves into the learning experience, facing the mysteries and the encryptions with near-endless energy. We were in a time of giving and the computer gave in return. No wonder that some of us can feel disenchanted, now that the bloom of excitement is past. The computer isn't giving anymore... and neither are we. Is this the very end of the relationship?

Hackers can have a lonely life, unless they share their work and their ideas, unless they communicate with other human beings who enjoy what they enjoy and appreciate the work at hand. This sharing takes some of the burden of giving from the computer and puts it back in the world of human beings where it belongs. Our theme for Volume VI —getting to know one another better— has a contribution to make if you would like to rekindle that old

from the Editor's Desk

flame. The challenge of writing articles is a sure way to "get serious" again. Knowing your effort will be read and appreciated by others might put some energy into those tired old bones. People who feel nourished by the computer experience are often those who tend to feel gratified by the problem solving aspect of their activities, and problem solving implies 'working for a purpose.' Rather than focusing on the computer, they try to see how it can be a tool to aid them in immediate needs, perhaps managing home finances or computing household or business data. (I have yet to see a good comprehensive program for home money management.) Using the computer in a new way, such as learning a new programming language, will broaden our interest to be sure. Different computer languages work to sharpen one's perception of the idea of programming, just as learning human languages increases one's perspective on the human experience as seen in differing cultures and histories.

Here are a few suggestions that might inspire your continuing pleasure at the keyboard. In this issue you will find some themes that need explanation in Colorcue—questions other readers have asked. Which one can you unravel for us? We are introducing a contest in this issue, and I hope our Colorcue readers in school user groups will take special notice. The pen plotter article may inspire you to look into a screen dump, in Basic or 8080 code, with four color display, and you can help many of us by explaining some simple programs in Forth, Pascal or the 'C' language.

Gary Dinsmore has a proposal for readers concerning a way to distribute their home-made software which might also add fuel to the new fire. Your participation is the key to all this and our guess is, that when you put some of your energy back into your computer education, your CCII will come through.

Jhn.

User Supported Software

* * * * * * * * * *

Gary Dinsmore
32695 Daisey Lane
Warren, OH 97053

I have been an active programmer in Basic and assembly language for the four years that I have had my Compucolor II. I have also done some Apple programming and I much prefer the Compucolor file control system to the Apple. I have flushed out most of the routines in the FCS and CRT ROMs. I have had a lot of help from the articles in Colorcue and Forum International from such authors as Myron Steffy, Capt. DeFrance Clarke, David Suits, and Dale Dewey. Now, with the help of Ben Barlow's Basic Tracer, I am listing and interpreting the Basic ROM.

You can imagine that I have a considerable investment in time and knowledge of the Compucolor computer. There is another side to the coin. At work, I recently purchased an new IBM PC with some 'super' programs like Lotus 1-2-3© and DATASTAR© by Micro-Pro. Each of these programs cost $400 or so and there are many more being offered by software houses because of the large market potential. I was able to obtain a demonstration of these programs by my dealer, using typical data that I supplied.

If our Compucolor software base could be kept active and progressive, the CCII would remain, even today, one of the better computers available. We need to keep pace with the advances in user-friendly, higher efficiency software products now available. Several software hackers writing for the CCII, including myself, have tried to sell directly to readers of Colorcue and Forum through advertising in these periodicals. I, at least, have not tried to obtain distribution through retailers so far. But there is a new and interesting marketing concept taking form in the large markets. Programs are sent free of charge, upon request, to prospective customers, on a trial basis—software owned and copyrighted by the author. The user of the program is asked to pay a license fee for using the program. The advantage is that one may try the program before spending money for it.

I know I am not alone in having purchased software, with formalized licensing agreements that I had to sign and return forthwith, or suffer unimaginable consequences, which has not performed as advertised. We have spent weeks working out bugs as they appeared. I have also purchased programs that have no hope of accomplishing the tasks they have set out to do. The old ISC Personal Data Base is a good example of that. On the other hand the Machine Language Degug package and FREDI are programs I would not like to be without.

Like many of you, I operate on a shoestring budget and there is no personal IBM PC in my near future, and without it, no new and powerful commercial software. The solution for us may be the concept of user-supported software. This concept includes the following features: distribution of the software is free; the author owns and copyrights the software; manuals and documentation are contained on the disk in an easily-printed form; the user receives this package and tries it, being on his honor to send a fee to the author if the program proves satisfactory. Under this program, users are free to copy and distribute the software to others with the understanding that these recipients are also subject to the same priveleges and obligations as the giver.

There are some problems. Theft is possible, but the protections against it are so tenuous and expensive to enforce as to make them ineffective. Follow-up support for the software can be complicated in some cases, and providing adequate documentation by only the contents of an ASCII file might be difficult. If owners modify and pass on a program, there will be many versions in circulation, making it awkward for the

author to give responsible assistance to purchasers. Nevertheless, these risks do not diminish the potential benefit of encouraging a new approach to distribution in our small CCII marketplace. We are clearly not going to find software support from larger businesses.

I propose to create a User Supported Software business, with encouragement to other programmers to register their software with me so I may distribute it, with my own, in the manner just described on an honor system basis. My service charge would be 10% of the collected sale value, payable at the time the fees were collected. I would try to provide a reasonably standard format and procedure for printing manuals from disk, and periodic update bulletins to purchasers to describe improvements, corrections, availability of more recent versions of their software, and promotional material on new products. Such a bulletin could be issued quarterly. The honor system would prevail at all stages of these transactions.

Will it work? Will users honor the author's right? I believe they will. The experiences of Frepost Computer, which distributes Freeware® for the IBM PC group of computers, receives about an 80% return on distributed programs, according to a recent article in PC magazine. If you have software and would like to try this user-supported approach, why not join me. Additional information may be found in my advertisement in this issue. □

BASE 2 RIBBONS: If materials received at the Colorcue office are an accurate indication, most Base 2 printer owners haven't had a ribbon change in several years. You may order ribbons for your printer from: BCCOMP-CO; 800 South 17, Box 246, Summersville, MO 65571. PHONE: 417-932-4196

Prices for new ribbons are 2 for $15 and 6 for $42. Payment may be by VISA, MASTERCARD, check, money order, and COD. They pay shipping on all prepaid orders and list in their catalog the following ambiguous line which may be a collection of type numbers you can use to identify your present cartridge: DIP81/82/84/85G.

A RETURN TO FCS

Tom Napier
12 Birch Street
Monsey, NY 10952

A recurring problem in assembly programming is the exit to FCS following program execution. The usual procedure is to save the FCS stack pointer at RUN time, then restore it at exit and use a RET to invoke FCS. This works fine if program execution is initiated by an FCS RUN command, but it fails, of course, if program execution is initiated by ESC T or ESC USER. Some software circumvents this problem by exiting to CRT mode (such as MLDP.) I have been looking for a way to exit to FCS without run-mode conditions —a software equivalent to ESC D.

Typing ESC D moves the program counter to a subroutine called ESCD in ROM beginning at 32C9H (v6.78 or 16FFH (v8.79, v9.80). This routine begins with MOV M,B so one can see that ESCD assumes a value already in HL. A little research shows that this value has to be the keyboard flag 81DFH if one is to continue by issueing an FCS command from the keyboard. Thus, setting HL to 81DFH and jumping to ESCD should restart FCS. There are two snags.

First, ESCD is actually a subroutine and needs to have a return address supplied; second, we don't want the stack pointer to wander all over the screen, as it will if the FCS stack is not kept under control. RUN saves the existing machine state on the stack, whereas ESC T, etc, does not. Investigation of the stack contents suggests that by setting the stack pointer to 8034H one could reclaim the conditions left by RUN by restoring from either RUN or an ESC sequence. This was the essence of my first attempt and it worked most of the time. However, sometimes I found myself in Basic with a syntax error message and other times I found myself back in the program I was trying to exit.

The solution is to ignore the existing stack contents entirely and start all over. The routine in Listing I starts FCS afresh every time it is used, with no error messages, no endlessly growing stack and no need to save the stack pointer on program entry. It even puts in a "wait for the user" input loop return address for ESCD to use.

The Basic ROM contains a routine similar to mine, beginning at 0564H, but it's not convenient to use since it expects to find 81DFH stored at 8200H which is where Basic puts it. Since 8200H is usually the first address of the user's program it's not the best place to store a pointer—not if one wants to reenter a program successfully (an unfortunate circumstance in FCS design; why does ESC T vector to 8200H and confuse Basic's pointer area?) Many happy returns. □

```
EXIT:    LXI    SP,8042H     ;Restart stack pointer*
         LXI     H,0038H     ;Vector to input loop
         PUSH    H           ;RET for ESCD
         LXI     H,81DFH     ;Keyboard flag
         JMP     ESCD        ;(Check text for address)

*Note - Restart stack pointer address for v9.80 is
8044H
```

ASSEMBLY LANGUAGE PROGRAMMING

Part XIV: ASCII, Masks, and BCD.

Joseph Norris
19 West Second Street
Moorestown, NJ 08057

We own the mixed blessing of four number systems in our computers, binary, hexadecimal, binary-coded decimal and the ASCII code, all of which come into play during ordinary computer operation. While all the necessary conversions are straightforward, they are not obvious; manipulations can be quite involved at best.

This article is a key to unlock that evasive door. (I hope you will find a "lady" and not a "tiger.") We will look at some relationships among the characters in the ASCII set and at the conversion of binary numbers into a form suitable for screen display.

There is magic residing in the 8080A logic instructions to make life easier and more interesting when using the elements of ASCII. This magic is potentiated by the instructions ANI and ORI and their ability to format selective filters (sometimes called "masks") through which binary data may be passed. The usefulness of such devices is apparent when relationships among elements of ASCII are examined. Look at the comparison between the binary forms of hexadecimal and ASCII for the numbers from 0 to 9.

Decimal #	HEX #	(Binary) #	ASCII #	Character	(Binary) #
0	00	0000 0000	48	"0"	0011 0000
1	01	0000 0001	49	"1"	0011 0001
2	02	0000 0010	50	"2"	0011 0010
3	03	0000 0011	51	"3"	0011 0011
4	04	0000 0100	52	"4"	0011 0100
5	05	0000 0101	53	"5"	0011 0101
6	06	0000 0110	54	"6"	0011 0110
7	07	0000 0111	55	"7"	0011 0111
8	08	0000 1000	56	"8"	0011 1000
9	09	0000 1001	57	"9"	0011 1001

The difference between the binary forms of these number sets is the presence of '0011' in the four 'most significant bits' of the binary ASCII code (bits 8-5). If we were to want a conversion from HEX to ASCII, all we would need to do is add '0011 0000' (48 decimal or 30H) to the binary number, as long as the number is less than 10 (0AH).

```
    05H = 0000 0101B =  5 Decimal
+   30H = 0011 0000B = 48 Decimal
    ---   ---------    --
    35H = 0011 0101B = 53 Decimal = "5"(ASCII)
```

If we transgress beyond 09H we will be disappointed:

```
    0AH = 0000 1010B = 10 Decimal
+   30H = 0011 0000B = 48 Decimal
    ---   ---------    --
    3AH = 0011 1010B = 58 Decimal = ":" (ASCII)
```

To make the reverse conversion, ASCII to HEX, we can subtract 30H—or its equivalent—from the ASCII number. Now take note that, when adding or subtracting numbers, both numbers must be expressed in the same number system; we cannot directly add 30H to 10 decimal and feel confident![1]

From our discussion one can see that the following routine will convert HEX to ASCII with ease, for numbers between 0 and 9:

```
MVI  A,05H    ;Place the number "5" into the accummulator.
ADI  30H      ;Increase it by 48 decimal
CALL LO       ;Print the ASCII symbol of the resultant
              ; sum (whose value is 35H.)
              ;A "5" will be printed on the screen. 35H
              ; will be in the accummulator. Note that any
              ; further arithmetic on the number in the
              ; accummulator will be inaccurate unless
              ; we subtract 30H to restore the
              ; proper Hex value to A.
```

There is another way to make this conversion, using the ORI instruction. This instruction, as do all the 8080A instructions, operates on the binary form of the data. Suppose we OR the numbers 30H and 05H:

```
Bit Number 8 7 6 5  4 3 2 1      Working with each bit at a
                                 time, we perform the OR
30H  =     0 0 1 1  0 0 0 0 B    operation which says, if
05H  =     0 0 0 0  0 1 0 1 B    either bit is a "1", make
           ---------------       the answer a "1"; if both
                                 bits are "1", make the
OR result  0 0 1 1  0 1 0 1 B    answer a "1"; if both bits
                                 are "0" then make the
                                 answer a "0."
```

ORI (num) is the OR-Immediate instruction. It offers no time advantage over the ADI instruction (both take 7 cycles) but it presents the conversion process in "logical" form and puts us in a good mental framework for other logic-based operations. While in this case ORing is the same as an addition, we might rather think of it as a conversion of form —inserting active bits 5 and 6.

Look now at the relationship between upper and lower case letters in the ASCII character set:

LETTER	ASCII #	BINARY	LETTER	ASCII #	BINARY
"A"	65	0100 0001	"a"	96	0110 0001
"B"	66	0100 0010	"b"	97	0110 0010
"C"	67	0100 0011	"c"	98	0110 0011

.... etc.

Here we see a changed 6th bit —that is, we have "added" 32 decimal, 20H, or 00100000B to convert upper case to lower case. "Subtracting" the same amount from lower case will produce upper case. (This is sometimes called 'stripping.') In the logic form of the process we use the ANI instruction, which ANDS one number with another, and looks like this:

```
Bit number 8 7 6 5  4 3 2 1      Working with each bit at a
                                  time, we perform the AND
98H  =     0 1 1 0  0 0 1 1      operation which says, if
XXH? =     1 1 0 1  1 1 1 1      both bits are "1" then
           -----------------      make the answer a "1",
                                  otherwise, make it a "0".
AND result 0 1 0 0  0 0 1 1
```

In this example I haven't even thought about the HEX value of the number I will use to AND with 98H. I have simply arranged my '1''s and '0''s to remove (''strip'') the 6th bit to convert to upper case. I am working only on the binary form of the data. Here is the assembler code to do the same thing:

```
MVI  A, 98H          ;ASCII "c", A = 0110 0011B
ANI  11011111B       ;Perform binary "strip."
                     ;A now holds 0100 0011B = "C"
```

In the byte following "ANI", above, I have arranged '0''s and '1''s to assure that only bits that are set in bit positions 1, 2, 3, 4, 5, 7, and 8 are 'passed through.'

I can also see to it that bit 8 —never used in ASCII— is always reset, to prevent non-printing characters from entering LO. To do this I "AND" my ASCII code with 0101111B, eliminating any 8th-bit data that might exist.

We summarize our use of ANI and ORI as follows: use ORI when you want to be certain a particular bit is set; use ANI when you want to be certain a particular bit is not set.

Returning to the numbers: what good is it to have such a limited conversion facility for only 9 numbers and "0"? Not much, in fact, if we are truly limited this way, but the 8080 instruction set allows us to express binary numbers in ASCII characters by first converting our data to binary-coded-decimal form (BCD). Here is the binary and BCD form of the numbers between 27 and 33:

DECIMAL	BINARY	BCD
27	0001 1011	0010 0111
28	0001 1100	0010 1000
29	0001 1101	0010 1001
30	0001 1110	0011 1010
31	0001 1111	0011 1011
32	0010 0000	0011 1100
33	0010 0001	0011 0011

For the number 27, in binary form, the bits are set in the usual powers of 2. In the BCD version, the first digit of 27, the '2', is expressed in binary by the four left-most bits (the "high nibble"), and the '7' is expressed in binary form by the four right-most bits (the "low nibble.") When using BCD form, the powers of 2 are rearranged like this:

BIT NUMBER	8	7	6	5	4	3	2	1	
PLACE VALUE	8	4	2	1	8	4	2	1	
SAMPLE CODE:	1	0	0	1	0	1	1	1	= 96 (D)

Each nibble of a BCD number will never contain a number greater than 9, because the computer routines are designed that way; therefore, 1001BCD is the highest code that any nibble will contain. It is also true that every BCD nibble can be converted to ASCII, since '0' through '9' is available in ASCII. For each decimal power of 10, in our decimal number to be converted, there must be a 4-bit BCD nibble somewhere in memory at the time of display. Two nibbles (the easiest to work with) can accommodate all numbers between 0 and 99. Three BCD nibbles will take us to 999, and so on.

The 8080 instruction set provides an instruction dedicated only to the process of converting 8-bit binary numbers into 2 groups of 4-bit BCD numbers. It is called DAA (Decimal-Adjust the Accummulator.) Here is an example:

```
MVI B, 00011001B          ;Place 25 (decimal) in B
MOV A,B                   ;Copy it into A
DAA                       ;Decimal-adjust A
MOV C,A                   ;Copy result into C

Contents of B = 00011001  Contents of C = 00100101
```

The number to be adjusted must be in the accummulator. We are not yet in a very good position, for if we attempted to print as ASCII what resides in the C register we would get the ASCII character for 37 which is "%" and not the "25" we expect. We must find a way to separate the 8-bit BCD code into two 4-bit nibbles and convert each nibble separately.

If we could move all 8 BCD bits into A and mask off the first four bits (that is, set them all to '0') we could convert, at least, the last four easily. We know how to cancel those first four bits—with an AND instruction.

Using the value of C, above:

```
MOV A,C                 ;Move the BCD number into A.
ANI 00001111B           ;Set high nibble to 0000
```

The contents of A is now [5] 0000 0101. Furthermore we must add the ASCII 'fudge factor'—00110000B to this result to convert it to the ASCII character for '5'; continuing -

```
ORI 00110000B           ;A now holds "00110101"
CALL LO                 ;Screen sees "5"
```

Now for the high nibble, lets put C back into A. We somehow have to get the four high bits to trade places with the four low bits so the accummulator looks like this:

```
0101 0010
```

and we can do that by rotating the bits in the accummulator four times, without passing through the carry bit.

```
MOV A,C                 ;Copy C back into A
                        ;A= 0010 0101
RRC                     ;Rotate once to the right,
                        ;A= 1001 0010
RRC                     ;Rotate again, A= 0100 1001
RRC                     ; and again,   A= 1010 0100
RRC                     ; and again,   A= 0101 0010

ANI 00001111B           ;Mask "high" nibble
                        ;A= 0000 0010
ORI 00110000B           ;Convert to ASCII
                        ;A= 0010 0010
CALL LO                 ;Print it! Screen sees "2".
```

Our only problem is that we have "52" on the screen instead of "25." We clearly have to reverse the order of our conversion to see that the high-order nibble gets printed first.

With this procedure we can now print in ASCII all the positive numbers between 0 and 99. Does it stop there? Of course not, but our routine becomes more complicated, because we must deal now with 16-bit and even 32-bit numbers, breaking them down into manageable size, adjusting them properly, and printing them. It would be good exercise to experiment with a program that will act as an adding machine for single digit addends, between 0 and 9, and for sums between 0 and 18, using a system of BCD conversions. ADD.SRC will do this. It is a "Mickey Mouse" program which illustrates the procedures we have been discussing. When you understand how it works, you might try to expand it to include double digit addends, for sums up to 99.

We need a single character input routine so we borrow again from David Suits' article in COLORCUE JUNE/JULY 1982, using just enough to get one character into the accummulator. A little error checking routine has been added for completeness. □

```
;ADD - testing ALP XIV: Simple ASCII
; to  binary conversion.
;Uses single key input routine.
;Inputs single digit numbers, 0-9,
; adds them and prints sum < 19
;Addresses shown for v8.79 & v9.80
;Addresses for v6.78 in parentheses.

KBCHAR  EQU     81FEH   ;(81FEH)
LO      EQU     17C8H   ;(3392H)
OSTR    EQU     182AH   ;(33F4H)

        ORG     8200H   ; - or whatever

START:  MVI     A,0C3H  ;Setup to CHRINT
        STA     81C5H
        LXI     H,CHRINT
        SHLD    81C6H
        MVI     A,1FH
        STA     81DFH

;****** MAIN PROGRAM ****

GO:     LXI     H,READY ;Clear screen, etc
        CALL    OSTR
        CALL    NUM1    ;Input first number
        CALL    NUM2    ;Input second number
        CALL    ANSWER  ;Add and print sum
        LXI     H,CONT  ;Hold screen display
        CALL    OSTR    ; for viewing
        MVI     C,0     ;Defeat error processing
        CALL    INPUT   ;Get "continue" keypress
        JMP     GO      ;Start again
```

---

[1] Your assembler will translate among number systems, of course, if you indicate what system each number is meant to be in. When working with members of the logic instruction set, it's meaningful to use the binary form of the numbers.

[2] The ANI and ORI instructions affect the flag bits, and, as such, are useful for presetting certain flags. Consult a good text to review this and other uses of all the logic instructions. Possibly the best entry-level text is Fernandez, Judi N. and Ashley, Ruth. Introduction to 8080/8085 Assembly Language Programming. NY: John Wiley and Sons, 1981. (I found a copy in an APPLE computer store some time ago, which was grateful to see it go, but not grateful enough to reduce the price—about $10.95.) If your local store doesn't have it, you can probably order it through any bookstore. The sources cited in my article in COLORCUE FEB/MAR 1983, p. 17 are also valuable.

```
;****** SUBROUTINES *****

NUM1:   ;INPUT FIRST NUMBER

        LXI     H,FIRST ;Print message and
        CALL    OSTR    ; position cursor
        MVI     C,1     ;Set correct error response
        CALL    INPUT   ;Get first number 0-9
        CALL    LO      ;Echo to screen as ASCII
        ANI     00001111B       ;Convert to binary
        MOV     D,A             ; and store
        RET

NUM2:   ;INPUT SECOND NUMBER

        LXI     H,SECOND
        CALL    OSTR
        MVI     C,2
        CALL    INPUT
        CALL    LO
        ANI     00001111B
        MOV     E,A
        RET

ANSWER: ;COMPUTE SUM, CONVERT AND PRINT

        MOV     A,E     ;Retrieve first number
        ADD     D       ;Add second to it
        DAA             ;Convert to BCD
        STA     TOTAL   ;Store sum in BCD form
        LXI     H,PLOT  ;Position cursor
        CALL    OSTR    ; for display
        LDA     TOTAL   ;Retrieve sum in BCD
        RRC             ;Rotate high bits into
        RRC             ; low position
        RRC
        RRC
        ANI     00001111B       ;Mask off 'high'
        CPI     00H             ;If value is "0" then
        JZ      SPACE           ; print space
        ORI     00110000B       ;Otherwise convert
        CALL    LO              ; to ASCII and print "10's"
NEXT:   LDA     TOTAL           ;Retrieve sum again
        ANI     00001111B       ;Mask off high bits
        ORI     00110000B       ;Convert low bits to
        CALL    LO              ; ASCII and print
        RET

SPACE:  ;PROVIDE LEADING SPACE FOR SUM < 10

        MVI     A,20H           ;Put ASCII space in A
        CALL    LO              ;Print it
        JMP     NEXT            ;Go back for 'ones' digit

INPUT:  ;GET SINGLE DIGIT ADDEND

        XRA     A               ;Zero A
        STA     KBCHAR          ;Clear KBCHAR
XX4:    LDA     KBCHAR          ;See if key pressed
        ORA     A               ;Set flags
        JZ      XX4             ;Key not pressed

        ;Check if keypress was "0" - "9"

        CPI     48      ;Equal to or greater than ASCII 0?
        JC      ERROR   ;No! Invalid key, process error
        CPI     58      ;Equal to or greater than ASCII 10?
        JNC     ERROR   ;Yes! Invalid key, process error
        RET

CHRINT: PUSH    PSW     ;Character interrupt routine
        XRA     A
        STA     81FFH
        POP     PSW
        RET

ERROR:  ;PRINT ERROR SIGN IN APPROPRIATE PLACE

        MOV     A,C     ;See whether 1st or 2nd number
        CPI     1
        JZ      ERROR1  ;It was first number
        CPI     2
        JZ      ERROR2  ;It was second number
        JMP     GO      ;It was "continue" signal

ERROR1: LXI     H,ER1   :Position cursor to first number
        CALL    OSTR    ; and print red "?"
        JMP     NUM1    ;Go back for legal entry

ERROR2: LXI     H,ER2   ;Position cursor for second number
        CALL    OSTR    ; and print red "?"
        JMP     NUM2    ;Go back for legal entry

;*****************************

;Clear screen, set green, erase page, small characters,
; page mode -

READY:  DB      6,2,12,27,24,15,3,8,6,'ADD'.SRC'
        DB      '  - Testing ALP XIV',239

;Position cursor for first entry -

FIRST:  DB      3,10,10,'FIRST NUMBER  )'
        DB      3,30,10,239
```

```
;Position cursor for second entry -

SECOND: DB      3,10,12,'SECOND NUMBER )'
        DB      3,30,12,239

;Format for printing of sum -

PLOT:   DB      3,28,13,'____'

        DB      3,10,15,'ANSWER
        DB      3,29,15,239


ER1:    DB      6,1,3,30,10,'?',6,2,239

ER2:    DB      6,1,3,30,12,'?',6,2,239


CONT:   DB      6,3,3,8,20,'PRESS ANY KEY TO '
        DB      'CONTINUE',6,0,3,65,20,239

;Temporary storage of sum. We need to preserve it since
; we are destroying a nibble with each mask -

TOTAL:  DS      1

        END     START
```

HELP WANTED: "I just purchased an Anchor Automation direct connect Bell 212A modem but I am unable to send a 'break' to a host computer, such as IBM or Honeywell. The subroutine I am using is -

```
BREAK: MVI  A,10
       OUT  4
       MVI  A,200   ;Setup wait routine
       CALL WAIT    ; for approx 200 mS
       MVI  A,8
       OUT  4
       RET
```

"I have a 300 Baud acoustic modem which works with this break routine. I have tried the break switch in Colorcue, March 1980, with no luck. Can someone offer some suggestions?" Chris Zerr. 14741 N.E. 31st Unit 1-C, Bellevue, WA 98007. (206) 882-1746. Compuserve 71445,1240.

* * * * * * * * * *

—SOURCEBOOK—

Last chance to get your materials in for the MAY/JUNE issue. See VOL VI No 1 for details. It will be a big one and possibly a little late, so be patient! Press date is May 30th.

We are accustomed to presenting the computer with a question and having it give back an answer. Here is a different kind of computer problem—given a set of required answers, have the computer derive a set of equations to produce them. This kind of problem solving is often required in higher mathematics and in scientific "modelling" of all kinds. The simple problem described here is not unlike asking a computer to write an integration equation that will fit some known facts in the form of data.

Test your wits with this one. Given the group of "answers" 1 through 9 inclusive, and any three different initial integers, each less than "5", have the computer print out an equation using each of the three given integers and the mathematical operators '+', '-', and '×' to produce each "answer" in the group. For example, given the integers A, B, and C, show equations in the form $A(\checkmark)B(\checkmark)C = X$, where $(\checkmark)$ is one of the operators '+', '-', or '×'. Your printout will look something like this .

$$A\ (\checkmark)\ B\ (\checkmark)\ C\ (\checkmark) = 1$$
$$A\ (\checkmark)\ B\ (\checkmark)\ C\ (\checkmark) = 2$$
$$A\ (\checkmark)\ B\ (\checkmark)\ C\ (\checkmark) = 3$$
$$\text{----------}$$
$$A\ (\checkmark)\ B\ (\checkmark)\ C\ (\checkmark) = 9$$

The integers and mathematical operators may be in any sequence, each used only once in a single equation. This problem has been presented elsewhere for examination on a number of occasions. It presents a fascinating challenge for programmers of all ages and experience.

To motivate you somewhat, Colorcue is having a contest. We will award two prizes, each a gift certificate for $35 , redeemable at Intelligent Computer Systems in Huntsville, Alabama. One award will be given to the winning entry by a student or group of students under the age of 19 years, and the other to the winning entry by an individual 19 years or older.

Entries must be in the form of a printed listing of an executable Basic program for the CCII in v6.78, v8.79 or v9.80-3, which will print the parameters and equations on the CRT and/or a line printer. (Note: it is not always possible to derive a valid equation for each answer 1-9, but your program should provide all possible answers for the three integers you have selected.) Entries will be evaluated by the Colorcue staff for neatness, compactness, appropriate use of color and graphics, and clarity of program annotation. Entries must be postmarked no later than June 30, 1984. The winning programs will be published in the SEP/OCT issue of Colorcue.

COMPILING BASIC

Part III

Peter Hiner
11 Penny Croft
Harpenden
Herts, AL5 2PD
ENGLAND

We have previously seen how variables and strings are handled by the Basic Interpreter and the FASBAS compiler. Now we move on to look at arrays.

In many respects numerical and string arrays are like variables and strings. Each location within a numerical array contains a 4 byte floating point value and each location within a string array contains references to the storage address and length of a string. Both types of array are stored together in a block of memory following the storage area for variables and strings. String arrays are differentiated by adding 80H to the second character of the name, just as for strings. Once the required location within an array has been accessed, its contents can be handled just as for an ordinary variable or string. So we will concentrate on the method for accessing the required location.

I will describe later how the compiler gives special treatment to single dimension numerical arrays, but for the moment I will deal with the normal method for accessing an array, using the single dimension array as the most simple example to clarify the principles.

The storage area for each array starts with two bytes containing the name of the array, followed by two bytes, indicating the number of bytes of storage space required by the complete array (this is used for skipping over the rest of the array when searching for the required array name.) Then there is one byte indicating the number of dimensions in the array, followed by as many pairs of bytes as necessary to indicate for each dimension the number of elements in that dimension. Each element within the array consists of 4 bytes of storage space, which contain the floating point value or the string references.

So if we take the simple case of a single dimension array such as A(10), which has the default size of 11 elements automatically allocated if we do not specify it otherwise in a DIM statement, we can calculate the number of bytes of memory it requires. The header requires 2 (name) + 2 (size) + 1 (number of dimensions) + 2 (number of elements in dimension), giving a total of seven bytes. Then the main body of the array will consist of eleven elements, A(0) through A(10), of four bytes each, giving 44 bytes, which makes a grand total of 51 bytes. A two—dimensional array B(10,10) would require 9 + 484 or 493 bytes and a three—dimensional array C(10,10,10) would require 11 + 5324 or 5335 bytes. So you can see that it is important to include DIM statements for multi—dimension arrays.

*R*eturning to our single dimension array, it is accessed by searching through the area of memory in which arrays are stored, skipping from header to header until the required array name is found. Then the contents of the bracket are evaluated to determine the number of the required element in the array. In the case of a single dimension array this is simple, as the element number is just multiplied by 4 (number of bytes per element) and used to skip over the required number of bytes and point to the element to be accessed. The contents of this element can now be treated as for an ordinary variable or string.

A two—dimensional array can be envisualized as consisting of rows and columns. If we want the element in the third row and fourth column, we need to multiply the column identifier by the number of rows in each column (for example 4 × 11 = 44) and then add the row identifier (44 + 3 = 47). Now we multiply by 4 and know that we must skip over 188 bytes to find the required element. You will note that this simple arithmetic conveniently takes account of the fact that the numbering of elements within a row or column starts from zero (did the egg come before the chicken?)

In a three—dimensional array we could consider the third dimension to be pages. Then we need to multiply the page identifier by the number of columns per page, before adding the column identifier and multiplying this total by the number of rows per column. Finally we add the row identifier and multiply the result by four (the number of bytes per element). You can have even more dimensions in an array, but access time is increased accordingly.

The fingerprint of history can be seen in the way FASBAS treats arrays. Originally I intended to handle only single dimension numerical arrays, and I included routines to minimize the access time. FASBAS allocates storage space at compilation time for single dimension numerical arrays. This storage space is included within the final compiled program and is labelled for direct access without searching for the array name. Of course the required element number must still be evaluated and converted from floating point to binary to allow access within the array, in much the same way as in Basic but without any extra activities such as checking that the element to be accessed really exists (e.g. that you are not trying to access an element 20 in a 5 element array).

*W*hen I later came round to adding multidimension and string arrays, I decided to let the interpreter deal with these in the normal way. FASBAS evaluates the contents of the bracket and generates instructions for the compiled program to push the results of evaluation onto stack and then to dive into the middle of the normal Basic Interpreter routine for handling arrays.

So multidimensional numerical arrays and all string arrays are stored dynamically (at run time) in the same manner and in the same memory area as for a normal Basic program. Access to them is still faster than in Basic because the compiled program speeds up evaluation of the contents of the bracket and because removal of single dimension numerical arrays will probably reduce search time.

The difference in treatment of single dimension numerical arrays from the others is primarily a result of my own unwillingness to produce special routines for multidimensional arrays. I do not think I would have gained much speed increase, and the routines would hav been long and complex. This decision can be justified by some benefits, but it has drawbacks as well.

On the plus side, allowing the interpreter to handle multidimension numerical arrays in the normal way permits implementation of the LOAD statement (and you can make a single dimension array into a multidimension array if you want to LOAD it). Also dynamic dimensioning (e.g. DIM A(x,y) instead of A(5,6)) is possible for multidimension numerical arrays and all string arrays.

*O*n the minus side, the different treatment caused extra complications in writing FASBAS and meant in particular that I had to include a check for number of dimensions during the first pass of the compiler, so that the program would know which type of treatment to apply to each array during the second pass. For the user it meant that I had to impose a requirement that multidimension numerical arrays must be dimensioned with a DIM statement, to help in indentifying them during the first pass of the compiler. In view of the number of bytes of memory space likely to by wasted by omitting DIM statements and allowing the default size to be allocated, this restriction is probably a good discipline.

We have now covered all the aspects of evaluating expressions, and a short reminder of 'comparison' statements will lead to my next topic. Evaluation of IF A = B THEN .... results in a value -1 in the Basic accumulator if 'true' and a value 0 if 'false'. The value -1 for 'true' is arbitrary, and any value other than zero would indicate 'true'. Therefore evaluation of IF A THEN ... gives a similar result, and means 'If A has a value other than 0 then ...'. The interpreter and compiler look at things the other way round, and say 'If the result is zero, then skip to the next line'.

*F*or the interpreter, skipping to the next line involves scanning through the rest of the current line until an end of line marker (null) is found, and then continuing with interpretation as normal. The compiler needs a label L.... for the next line so that it can generate an instruction JZ L.... . The compiler does not yet know what the next line number will be (and hunting back and forth through the Basic program would be complicated), so it generates a label based on the current line number with a suffix A (in line 100 it would generate JZ L100A). It sets a flag as a reminder that when it starts compiling the next line of Basic it must generate the label L100A: in addition to the normal label (e.g. L110:).

So the compiled version of
100 IF A = B GOTO 500
would be something like this:

```
L100:
LXI     H, VA    ;Point to variable A
CALL    .....    ;Move VA to accumulator
CALL    .....    ;Push VA onto stack
LXI     H,VB     ;Point to variable B
CALL    .....    ;Move VB to accumulator
POP     B        ;POP VA from stack
POP     D        ;To registers BC & DE
MVI     A,2      ;Def type of comparison
CALL    .....    ;Compare VA with VB
CALL    .....    ;Test result for 0 (false)
JZ      L100A    ;Skip to next line if false
JMP     L500     ;GOTO LINE 500 if true

L100A:
L110 :  .............   ; Etc.
```

(Please note that this and other examples of compiled programs may not be quite the same as the real thing, because subroutines in the run time library are often used to reduce program length or to perform functions like setting flags or sorting out the stack. The examples are intended to explain the principles.)

PRINTER SPOOLER: DATA EXCHANGE/64K is a spooler with serial or parallel in to 64K memory buffer to serial or parallel out. It will feed any peripheral, but is usually used to feed data to a printer at high speed, thus freeing your computer for other tasks without waiting for the printout to finish. Dip switches select input and output protocols which are independant. That means you can input at one Baud rate and output at another between 50 and 19200 Baud. An unlimited number of copies may be made from the buffer contents, and a panel switch clears the buffer for new data. One serial and one parallel cable is included, along with standard connectors for the other ports. RDY/BSY (DTR), Xon/Xoff, and ETX/ACK handshaking are all supported. List price is $339. Note that this unit performs the same function as the serial to parallel converter reviewed in this issue, which means you are paying an additional $245 for the 64K buffer. That's a fair price. Order from: Antex Data Systems, 2630 California Street, Mountain View, CA 94040. Phone: 415-941-7914.

In the above example there is an instruction MVI A, 2 which defines the comparison, but any value from 1 to 6 could be inserted to define each of the positive combinations of the 'less than', 'equals', and 'greater than' symbols.

The technique of generating labels with a suffix is used in other situations by the compiler. When the compiler meets a string in quotaton marks, it must label it and must include it somewhere in the compiled program. One possibility would be to hold all quotations in store until the end, but that might use up a lot of memory, so the compiler inserts them immediately into the body of the compiled program and jumps over them. The compiled version of PRINT "HELLO" would look like this:

```
        LXI     H,Q99   ;Point to string
        JMP     Q99A    ;Jump over it
Q99:    DB      '"HELLO";'
Q99A:   CALL    .....   ;Print it
        CALL    .....   ;Print a (cr)
```

The label Q99 means it is quote number 99, and the compiler simply numbers Q labels in sequence to ensure that there is no duplication of labels. The DB characters are an instruction to the Assembler to treat the rest of the line as data rather than instructions, and the single quotation marks (') indicate that the data between them is to be assembled as ASCII characters just as shown. So the assembled version will be "HELLO"; and this looks more like a bit of a Basic statement.

We are going to borrow a subroutine from the Basic interpreter to print the string, and we need to insure that it hands back control to our compiled program when it has finished. So we put double quotation marks (") at each end of the string as in normal Basic and then a semicolon (;) to prevent automatic generation of a carriage return when the interpreter meets the colon (:), which we have included to cause a return at the end of the statement. It may seem a bit odd that we prevent automatic generation of a carriage return, but then include a statement to include a carriage return.The answer is that this is the result of a standardized approach to PRINT statements in the compiler. It has to handle printing of other strings and variables, and does not make a decision about whether to generate a carriage return until it has returned from the EVAL subroutine to the PRINT routine, by which time it has already completed generation of the Q99 line and the label Q99A:.

The other situation in which the compiler uses labels with a suffix is in handling FOR...NEXT loops. Implementation of these loops was quite tricky and it was not until VER 12.21 that I worked out how to remove the limitations on premature exit from loops.

The principle governing a FOR...NEXT loop is that during implementation of the FOR statement a block of information is put on the stack, and when the NEXT statement is executed this information is read from the stack. If the loop is to be terminated, the information on stack is deleted, but otherwise it is left on stack and the program loops back to the statement following the FOR statement.

The information put on stack by the interpreter is slightly different from a compiled program (for example, the interpreter needs to store the line number in case it has to print an error message). However the principles are similar and I will list only the compiled information, in the order in which it would be read from stack when a NEXT statement is executed.

```
2 bytes - 'FOR' loop indicator
2 bytes - Address of variable
          controlling loop
2 bytes - Positive/negative
          step indicator
4 bytes - Step value (floating
          point)
4 bytes - Loop terminating
          value (floating point)
2 bytes - Loop return address
```

Taking the above information in order, the FOR loop indicator is an extra complication to deal with premature exits from loops, and I will discuss this in a moment. The address of the variable controlling the loop (e.g. VI in a loop FOR I = 1 TO 10) is required so that its value can be incremented or decremented by the step value and then compared with the terminating value, to determine whether to exit from the loop. It is also used to dealing with premature exits.

The positive/negative step indicator is used to determine the type of comparison to be made with the terminating value('greater than' if counting up, or 'less than' if counting down). The step value and terminating value are self—explanatory. The loop return address points to the statement following FOR I = 1 TO 10, and is generated by the compiler using a variation of the line number plus suffix technique. Because there might be more than one FOR statement in a line, and the loop return address must point to the next statement rather than to the next line, the compiler generates a double suffix (e.g. in line 120 it would generate L120A0 for the first FOR statement, L120A1 for the second, and so on.)

Premature exit from a loop is not good practice, but the interpreter permits it and therefore many people do it. In VER 12.21 of FASBAS I have included additional routines to cope with the additional stack growth (16 bytes of garbage for every premature exit) and the possibility that when a subsequent NEXT or RETURN statement is executed, the wrong information will be on top of the stack.

Every time a FOR statement is executed a check is performed to see whether the information on top of the stack relates to a FOR loop (i.e. if there is a FOR loop indicator on top) and, if so, whether it is a loop controlled by the same variable as the new FOR loop (in which case the previous data is to be removed from stack). The check is repeated down the stack until either some other data or the bottom of the stack is found.

When a simple NEXT statement is executed, a check is performed that there is a FOR loop indicator on top of the stack (if not, a Basic NF error message is given). A statement such as NEXT I will also cause a check that the FOR loop data refers to a loop controlled by

INTELLIGENT COMPUTER SYSTEMS INC.

- YESTERDAY -

There Was a Certain Kind of Person
Who Bought a Compucolor Computer

- TODAY -

There's the Same Kind of Person
Who Buys a **Morrow** Business Computer

You are the kind of person who doesn't follow the crowd - in business, or away from it. You've succeeded by making your own decisions.

And when it comes to a decision on computers, you know that you don't have to pay a lot of money to get a lot of computer. Morrow knows that too. Morrow is building computers for people who demand value.

- TOMORROW -

Don't Wait Till Tomorrow
Switch to **Morrow** Today

MORROW MD2	CP/M, with 2 single sided drives, each 190 K, incl. Wordstar	
MD2	2 ss diskdrives, Wordstar	US$ 995
MD2	incl MDT60 monitor, Wordstar	US$ 1395
MD2	incl. monitor, Wordstar and MP100 daisywheel printer	US$ 1895
MORROW MD3	CP/M, with 2 double sided diskdrives, each 386K, complete set of business software with Wordstar, Bookkeeper, Data Base, Checkbook, etc.	
MD3	2 dd diskdrives, software	US$ 1495
MD3	incl. MDT60 monitor, software	US$ 1895
MD3	incl, monitor, software and MP100	US$ 2395
MORROW MD11	complete system with 11 megabyte hard disk drive and 1 floppy, high resolution graphics monitor, and set of business software	
MD11	11 megabyte system	US$ 2645
MD11	system with MP100 daisywheel printer	US$ 3095

ORDER YOUR NEW MORROW SYSTEM FROM INTELLIGENT COMPUTER SYSTEM
12117 COMANCHE TRAIL, HUNTSVILLE, AL 35803, PHONE 205-881-3800

We still carry a complete line of Compucolor and Intecolor 3651 software.
Ask for catalog.

VISA, MASTER CHARGE AND AMERICAN EXPRESS ACCEPTED

INTELLIGENT COMPUTER SYSTEMS INC.

12117 COMANCHE TRAIL • HUNTSVILLE AL 35803 • PHONE 205-881-3800

I and, if not, the top data will be removed from stack and the checks repeated down the stack.

When a RETURN statement is executed the data on top of the stack is checked to see if it is a FOR loop indicator. The compiled program uses 8100H as the FOR loop indicator and this could not be mistaken for a RETURN address, so it would cause sixteen bytes to be removed from stack, and the check to be repeated. The interpreter uses a slightly different technique here, as it uses the single byte token for FOR (81H) and RETURN (8DH) to indicate the type of data on the stack, but the effect is the same.

The final part of this series of articles will deal with the remaining Basic commands and how to make a machine code program look like Basic. □

INTECOLOR SERVICE DEPARTMENT: Service facilities for the CCII and other Intecolor Corporation products has been moved to 5965 Peach Tree Corners East, Norcross, GA 30071. For the time being, at least, you may telephone the service department at (404) 449-5961. Unhappily, there is a dearth of service personnel with experience in servicing the CCII. The current charge for service is $150, with a 30 day warranty. It is best to send the entire computer back for service. If you attempt to save money by sending only the logic board, for instance, and problems multiply when it comes back, only logic board work is under warranty.

COMPUCOLOR MEETS

This article is dedicated to the late Myron T. Steffy whose tireless efforts provided us encouragement and with help when none was available elsewhere. His work on the Morrow project stands as a testimony to his unselfish concern for his fellow Compucolor users.

With hardware and software support dwindling, one hears more and more about the next direction for the Compucolor user. Since the CCII does not have much compatibility with other computers, many are making a 'hard' transition to another machine. But we would regret losing our software and our heavy investment in time and money—is it all lost or must we just let the world pass us by? Frankly, I do not want to give up my Compucolor. It is a good machine and I have yet to fully exploit all of its potential. If it ever goes "boom", I'd like to replace it with an Intecolor 3651 or whatever.

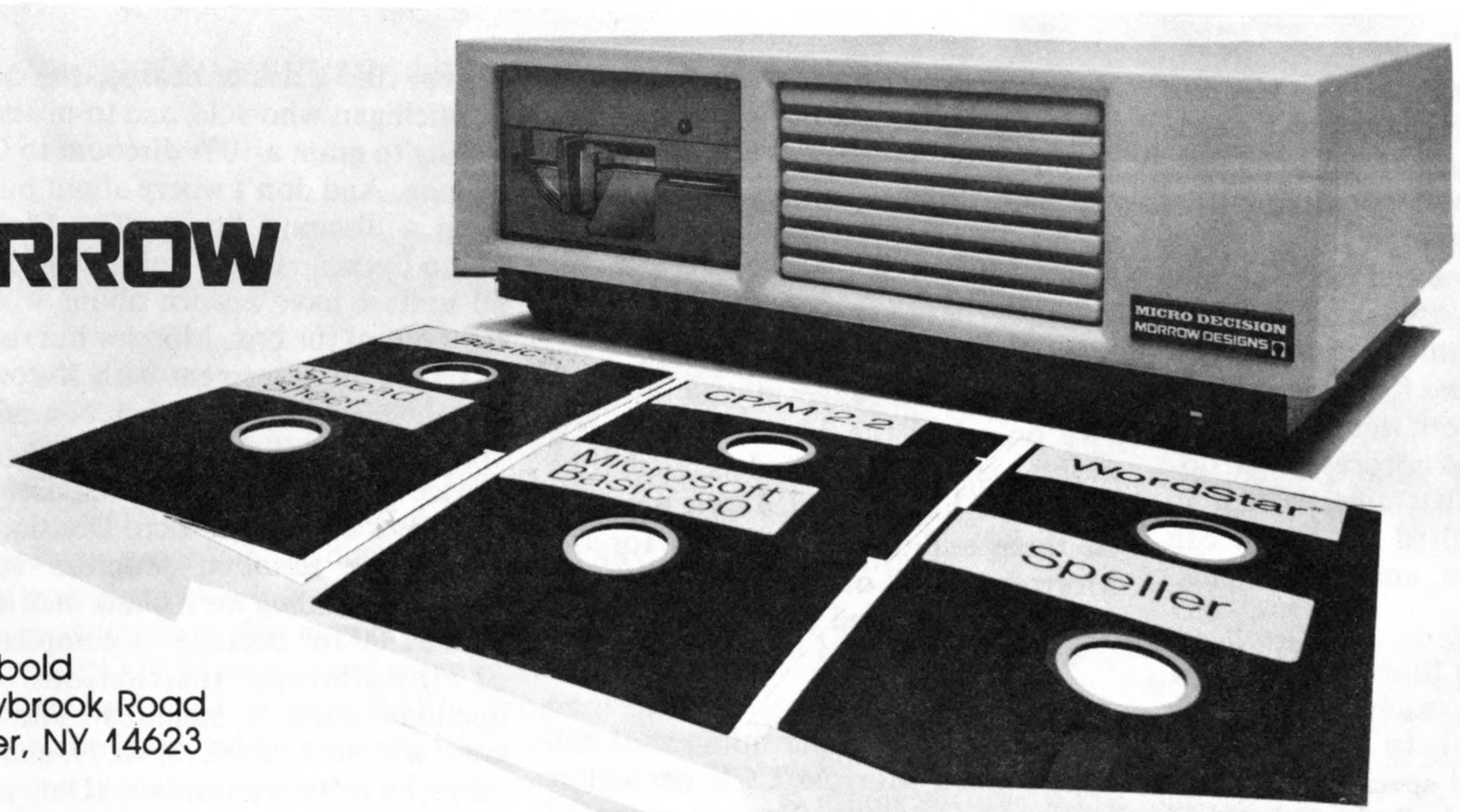
And then, what do we do about adding new software to our libraries? I have always planned to write most of my own materials, but there are programs which I would like to have that are not available for the CCII.

Enter —the Morrow Micro Decision! Tom Devlin, who had been looking for ways to expand the CCII easily, discovered the Morrow first and recognized its potential as an adjunct to the Compucolor computer. What is the Morrow Micro Decision? Simply put, it's a Z80A-based machine with 64K of RAM and full CP/M capability. It's compact, being scarcely larger than an Epson printer. It comes with 1 or 2 disk drives (200K each) and a large software package —CP/M 2.2, WordStar (word processor), a spelling checker, Microsoft Basic-80, and more. The software alone would cost over $1500 if purchased separately. The CP/M operating system provided includes a line editor and assembler. The assembler will recognize and assemble 8080 mnemonics. WordStar may be used as a screen editor for assembly language source development.

The best part of all is that the dual drive Morrow costs only $1000 (less terminal), including the software mentioned above. That's a fabulous deal! Notice that the Morrow is available less terminal, so Tom Devlin's idea was to use the CCII as the terminal. The Morrow is very easy to connect to the CCII through the RS232 port. The cable you may be using for a printer or modem connection will do. (See Ben Barlow's article in Colorcue, AUG/SEPT, 1981. Ed.)

There were a few problems initially. Because the CCII and Morrow speak different control codes, merely connecting the two units together produced unpredictable results. Tom Devlin began writing a terminal program so the two computers could communicate effectively and efficiently.

About this time Myron Steffy bought a Morrow, but in the time between Tom Devlin's purchase and Myron Steffy's purchase Morrow had made some changes and there were a few more headaches to do away with. In very short order, Tom's original program was extended to accommodate these dif-



Rick Taubold
197 Hollybrook Road
Rochester, NY 14623

ferences. Slowly, this masterpiece of programming took on the polish that seems to have been Myron Steffy's trademark. What we now have is a fully operational system, with a lot of Myron's "special" touches that make the whole system a joy to use. The terminal program source code is very well commented and easy to modify should further changes be needed.

But the program does far more than just provide a link between the Morrow and CCII. Myron added such features as single key commands for WordStar and the ability to transfer files from one computer to another, using WordStar on the Morrow and Comp-U-Writer or a screen editor on the CCII. Even Basic files may be transferred. By using the BASSRC program on the FREDI disk (or from some other source), the resulting SRC file can be sent to the Morrow through WordStar and saved on disk from there. One nice feature of BASIC-80 is that it can read such text files directly, eliminating the need for reconversion. While the transfer is slow (4800 Baud), I have successfully transferred two Basic files, one of which was over 26K bytes long. It took over 15 minutes, but it got there.

Microsoft BASIC-80 (MBASIC) which comes with the Morrow and runs under CP/M is a later version of the same Basic used in the Compucolor. BASIC-80 includes many more features than we have on the CCII. The losses are the lack of a graphics facility and the fact that the Morrow is "colorblind." This Basic will read text files directly, making program transfers from the CCII straightforward.

Most of you have heard of CP/M. For those unfamiliar with it, CP/M is just another operating system not unlike our FCS. While it is somewhat less user friendly, it has several advantages over FCS. First of all, it is a universal system, essentially independent of hardware, capable of running on many different computers. Its second advantage is that there are many, many programs written to operate under this widespread operating system. Thirdly, it is a RAM-based system, which means it must be loaded from disk each time it is to be used, and which also means it may be modified for special applications if desired. There have been many times CCII users have wanted to modify FCS (and some have done so) but this means making and installing new ROM chips—not a simple task.

One disadvantage of using CP/M on the CCII is that CP/M uses an 80-column screen display. The number of lines on the CCII are sufficient but the columns can cause difficulty. WordStar can be configured for a 64-column screen, but some other programs won't look right and a few CP/M programs simply require the 80-column screen to operate. For the latter case, the operator is left with no choice but to buy a proper terminal, usually in the price range of $600. The Micro Decision may be configured to operate with a wide variety of terminal types.

Another disadvantage of the Morrow system is that one cannot POKE things into screen memory as can be done on the CCII. The reason for this is that terminals lack memory-mapped video and only cursor addressing is available. This is not, however, a serious disadvantage. It may be possible to change the terminal program to overcome this deficiency for some applications.

What I'm saying is that the Morrow is probably the best thing to happen to us in a long time. You can still have all the features of your CCII plus access to an extended software library at a very reasonable cost. The Morrow Micro Decision is a sweet little computer. Its microprocessor, the Z80A, is twice as fast as our 80800A. Morrow provides you with a Micro-menu system, an "English" equivalent of CP/M commands written in PILOT, to help you along until you learn the native and more esoteric commands of CP/M. Morrow's own version of the PILOT programming language is included. The system comes with a stack of manuals, one for the computer itself, and one for each of the software packages. Most are well written. Only the CP/M manual (and that's Digital Research's doing and not Morrow's) is cryptic in places. In short, Morrow has done their best to provide an easy-to-use, yet powerful, system for a very reasonable price.

As an added feature, the Morrow allows you to read Osborne and Xerox 820 disk formats. I am seeing an increasing number of suppliers carrying programs directly formatted for the Morrow, but the Osborne and Xerox format capabilities almost insure availability of any CP/M program on the market. Morrow also provides for the reading of IBM disk formats, but this is less useful since the IBM microprocessor is different and only programs written in Basic could be utilized. Still, one can read the disk format, and this is a plus.

All in all, I have found the Morrow to be an interesting machine. Although CP/M is less friendly to use than FCS, Morrow has added special touches to remedy this situation. Perhaps the most interesting feature is the ability to get by with only one disk drive. Normally, CP/M won't allow this, but it does work here. I recommend that the two-drive computer be purchased, however. As it turns out, most CP/M commands are very similar to those of FCS, so it's not too hard to learn, and there are many books on CP/M available for tutorial help.

Consider now that you can have Pascal and C languages at your disposal. I've only had the Morrow for a few months and have not had as much time as I would like to use it. I have purchased three additional programs for the Morrow and all of them seem to work well. The increased disk storage alone is worth the price.

If we can add to our numbers, I will start a small Morrow/CCII newsletter section in Colorcue. There is potential here to learn more about the CCII and to learn CP/M as well. I'd like to hope that other CCII users will agree. If you are interested in buying a Morrow and cannot find a dealer nearby, the dealer in Michigan who sold one to me is still willing to grant a 10% discount to CCII owners. And don't worry about buying from a discount house. The Morrow Micro Decision is well-built, rugged, and all units I have known about worked right out of the box. Morrow has recently signed an agreement with Xerox for nationwide servicing of their computers. Wherever Xerox service exists, Morrow service is there also. Any CCII user who purchases a Morrow Micro Decision can obtain the terminal program at no charge by sending me a blank disk along with $2.00 for postage. A complete set of installation instructions and documentation is included. The program will work on v6.78, v8.79, and also on v9.80 software systems. (Only a few addresses must be changed, with a debug program, for v9.80 operation.) □

* * * * *

SUGGESTIONS FOR ARTICLES

Listed here are some topics that readers have expressed an interest in learning more about. Perhaps you can help.

1. Assume a reader is going to convert from COMPUCOLOR to another computer and CPU. Tell specifically how he can proceed to transfer CCII files to the new computer's files. Tell what he cannot do! Include BASIC programs, assembly programs and data files (RND to SRC, etc.) What software is available and from where, at what cost. Describe modifications that may be necessary such as conversion from PLOT sequences and using search and replace functions to perform this on a BASIC program in SRC form. Talk about the BASIC IN and OUT commands and the problems that may be encountered in converting these to other BASICS.

2. Describe how a reader can perform surgery on a failed computer. There are many failures that can at least be diagnosed by the user (perhaps with help from his friends with test equipment) so he knows what part is a likely suspect for the failure and can be replaced without a fruitless and expensive trip to the factory. What sources of repair are still available, prices, etc.

3. Continue discussion of disk routines in assembly: simulating random files, the READ BLOCK and WRITE BLOCK routines; special formats for disks (those without directories.)

4. Operation of peripheral devices such as plotters through the serial port in assembly.

5. Experiences with your new computer, or intended new computer. What are you looking for, what can you do without. How are you making these decisions. What have you looked at. What will you miss and not miss about the CCII.

6. Discuss Compucolor flags and their operation.

7. Program jumps using the ESC and CALL sequences. Believe it or not, many readers are still completely baffled by this technique in spite of the many articles that have already appeared. As an example, few people realize that ESC USER can call many different starting places within a single program simply by putting different JMP instructions in the associated memory location at different times.

8. Talk about the problems and techniques of computer games in assembly. Analyze a game like PAC-MAN. How do the little creatures 'know' where to go to gobble up! How do they 'know' what the maze pattern is. Design a creature using more than one plot block and show (in assembly language) how to move him around the screen with the joystick.

9. Think of a way to present an illuminating review of currently available CCII games written in assembly. Give sources and cost. Many readers see games mentioned but don't want to buy games in BASIC (the pig-in-a-poke syndrome).

10. Review some interesting utility software and describe exactly what it can do. I am thinking not so much about software already reviewed extensively, such as 'THE' Basic Editor, screen editors and assemblers, but more esoteric programs such as WISE and IDA.

11. Talk about COM files. How do they differ from PRG? Can the user create them? What's involved?

12. Discuss passing variables back and forth from BASIC to machine language.

13. Describe specifically how assembly routines can find and use specific variables and array members from BASIC and, perhaps, perform arithmetic operations on them and place results back into the array slot. This would lead to an assembly routine to speed up the recent COLORCUE article on CAD-CAM simulation.

14. Write a proposal for a standard communication procedure through the CompuServe network for CCII users (Colorcue can't be around forever, you know). There is evidence that readers want more contact but aren't sure how to get started. As we get smaller in numbers, increased avenues of communication are essential to keep the spirit alive. We can plan a network now to act as a 'last ditch' clearing house when other means no longer exist.

15. Discuss advanced BASIC commands such as the logical commands, defining functions, limited values of variables, such as D = (G greater than 54), etc.

16. Write on one of the many, many topics I have not included but that you just though of!

If you have hesitations about your time or ability to complete a finished article, we will perform a rewrite for you and submit the final draft for your approval before publication.

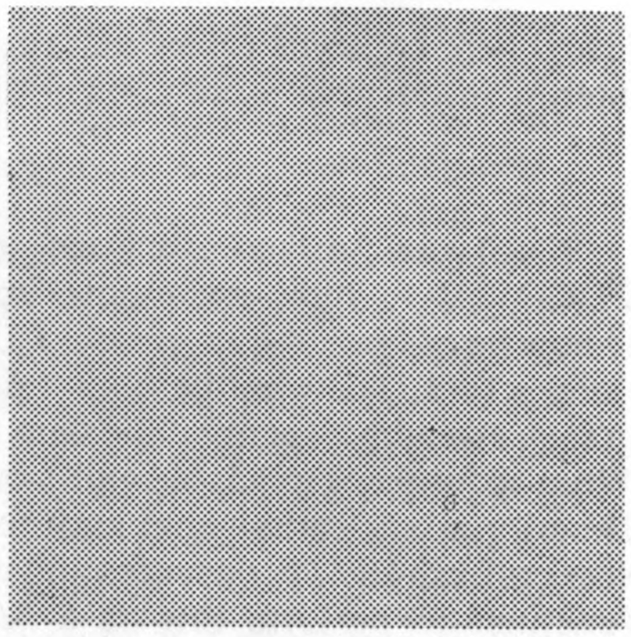
If there are line drawings, charts, graphs, etc, needed for your article and you do not want to make elegant renditions of these, we will prepare final copy from your complete and legible sketches. □

A Note On Writing Structured Algorithms

Charles H. Gould
317 Cocoa Avenue
Indialantic, FL 32903

I find that writing the logic of a program in algorithmic form helps me to keep the logic straight. After this step, coding the logic into Basic is straightforward (although frustrating). The use of Pascal, or Pascal-like, algorithmic language is useful for this purpose, and helps me 'structure' the program. For this purpose, I have used the technique illustrated in the listing. My thanks to David Suits for the intruiging program in Colorcue, June/July 1983, which I understood much better after rewriting it in Pascal. ('THE' Basic Editor made possible the Basic line indentations in the listing.)

```
100 REM   A PASCAL TO BASIC EXPERIMENT, CHG, 7JAN84
103 REM
105 REM   THE METHOD - First insert REM at start of each
110 REM                line, then write Pascal line.
115 REM               Next, translate each line into one
120 REM               or more lines of Basic on
125 REM               interleaving lines.
130 REM
132 REM
135 REM       -------Program Transformer-------
136 REM
137 REM
138 REM       **PASCAL**                  **BASIC**
139 REM
140 REM   VAR N,COUNT : INTEGER;
142                                       REM N=0, COUNT = 0
145 REM   BEGIN
148                                       REM
150 REM   WRITE('ENTER INTEGER ');
155                                       PRINT "ENTER INTEGER ";
160 REM   READLN(N);
165                                       INPUT N
170 REM   WHILE N <>  DO BEGIN
175                                       IF N=1 THEN 245
180 REM   IF ODD(N)
185                                       IF N/2=INT(N/2) THEN 205
190 REM   THEN N : = 3*N+1
195                                       N=3*N+1 : GOTO 215
200 REM   ELSE N : = N DIV 2;
205                                       N=N/2
210 REM   COUNT  : = COUNT+1;
215                                       COUNT=COUNT+1
220 REM   WRITE(N);
225                                       PRINT N
230 REM   END; (* WHILE *)
235                                       GOTO 175
240 REM   WRITELN;
245                                       PRINT
250 REM   WRITELN(COUNT,' STEPS');
255                                       PRINT COUNT;" STEPS"
260 REM   END.
265                                       END
```



MICROTELEMETRY MODULE: from Software Science. Works through the RS232 interface as an acquisition and control system, with 8 channel analog to digital and digital to analog I/O, decimal ASCII software commands and responses. 12 programmable digital output lines. Read data from switches, sensors, whatever. A safe interface for the beginner. Forget the problems of interfacing with the 50 pin bus! $249. Call (513) 561-2060 or write Software Science at PO Box 44232, Cincinnati, OH 45244.

EIA CABLES: with AMP connectors and housings, 20 conductors with moveable pins, male to male. Minimum order is five pieces: 5 feet at $5.00, 10 feet at $10.00, 15 feet at $12.00. The Printer Works. 1961 Alpine Way, Hayward, CA 94545.

NATIONAL 8073 Single Board Microcomputer: This chip contains National's tiny-Basic in ROM. Full 64K addressing, on-board EPROM burner and RAM with battery backup. Communicates through RS232 port at variable Baud rates. An ideal interface for the hardware experimentor. $156 fully populated, less power supply (+5 and -12 Volts DC.) EPROM burner and battery backup are additional cost options. Bare board available for $29.95. (Colorcue has one running a small N-gauge railroad.) It's called the MINI-73. Write to D.J.T. Electronics. 81 Orange Street, Sorrento, FL 32776.

CyPHEr

W. S. Whilly
(Address unknown)

```
                ;CYPHER: Encode from a table. By W. S.
                ; Willy for all his friends. v8.79 &
                ; v9.80 addresses are shown. v6.78 in
                ; parentheses. Unlike most things, this
                ; program is useless without its flaws.

(81FE)          KBCHAR  EQU  81FEH ;RAM
(17C8)          LO      EQU  17C8H ;(3392H)
(182A)          OSTR    EQU  182AH ;(33F4H)

0000 (8200)             ORG     8200H ;or wherever

8200 00       CYPHER: NOP     ;Fun to come!
8201 00               NOP
8202 00               NOP
8203 212583           LXI     H,CLR   ;Set up screen
8206 CD2A18           CALL    OSTR
8209 21D782           LXI     H,BXDSP ; and things.
820C CD2A18           CALL    OSTR
820F 212083           LXI     H,POS+1 ;Cursor X in
8212 3600             MVI     M,0     ; POS string=0
8214 23               INX     H
8215 360C             MVI     M,12    ;Cursor Y=12
8217 0E00             MVI     C,0
8219 212383           LXI     H,POS+4
821C 3602             MVI     M,2     ;Color POS green!

821E 3EC3             MVI     A,0C3H  ;CHRINT setup
8220 32C581           STA     81C5H   ; also from
8223 21A782           LXI     H,CHRINT
8226 22C681           SHLD    81C6H   ; Colorcue
8229 3E1F             MVI     A,1FH
822B 32DF81           STA     81DFH
822E C35E82           JMP     MAIN

                ;Main program:

8231 (825E)             ORG     $+002DH ;More fun 2 come!

825E 79       MAIN:     MOV     A,C     ;Count characters
825F FE40               CPI     64      ; entered, =64 ?
8261 CA0082             JZ      CYPHER  ; Yep! Go away
8264 CDBA82             CALL    INPUT1  ; NOP! Get next
8267 214884             LXI     H,CODE  ;Refer to table
826A BE       AGAIN:    CMP     M       ;Are chars alike?
826B CA7382             JZ      PRINT   ; Yes! Code it
826E 23                 INX     H       ;No! Index to
826F 23                 INX     H       ; next ref char
8270 C36A82             JMP     AGAIN   ; and try again

                ;Subroutines:

8273 E5       PRINT:    PUSH    H       ;Ready to print
8274 F5                 PUSH    PSW     ; so save regs
8275 212383             LXI     H,POS+4 ; & change color
8278 3602               MVI     M,2     ; to print input
827A 211F83             LXI     H,POS   ;Position cursor
827D CD2A18             CALL    OSTR
8280 F1                 POP     PSW     ;Restore A
8281 CDC817             CALL    LO      ;Print char
8284 212183             LXI     H,POS+2 ;Change cursor Y
8287 3612               MVI     M,18
8289 23                 INX     H       ; and
828A 23                 INX     H       ; color to print
828B 3601               MVI     M,1     ; coded char
828D 211F83             LXI     H,POS
8290 CD2A18             CALL    OSTR    ;re-pos cursor
8293 E1                 POP     H       ;Restore tble ptr
8294 23                 INX     H       ; index to code
```

[Editor's Note: This program was submitted some time ago by W. S. Whilly, with no return address. We didn't feel clear about publishing it because of the seemingly erratic format. However, a rather abusive letter from Mr. Whilly, who is incensed that we ask repeatedly for material but fail to print what we get, has prompted us to have second thoughts. (There was no return address on the second correspondence either.) Mr. Whilly goes on to explain that he does spell and program on a par with the rest of us, and that we must simply have faith in his program format. The most recent correspondence implies that Mr. Whilly means to use CYPHER as a medium for demonstrating the use of debug software in future articles. Since we have only the options of printing this or subjecting ourselves to a rash of abusive letters, we have chosen the former, and we extend our apologies to readers who may be offended by some of the material. Mr. Whilly has advised that if the initial origin of his program is changed, a hard-copy printout, with assembler addresses, must be made to utilize his subsequent material.]

```
8295 7E              MOV     A,M     ; move it to A
8296 CDC817          CALL    LO      ; and print
8299 212083          LXI     H,POS+1 ;Move cursor X
829C 34              INR     M       ; 1 space ->
829D 23              INX     H       ;Set cursor Y
829E 360C            MVI     M,12    ; to input line
82A0 0C              INR     C       ;Show -1 char
82A1 C35E82          JMP     MAIN    ;Back for more

82A4 C3A482          JMP     $       ; & more fun

82A7 F5      CHRINT: PUSH    PSW     ;From Colorcue
82A8 AF              XRA     A
82A9 32FF81          STA     81FFH
82AC F1              POP     PSW
82AD C9              RET

82AE AF      GTCHA:  XRA     A       ;Kbrd 'get' rout.
82AF 32FE81          STA     KBCHAR  ; from Colorcue
82B2 3AFE81  XX4:    LDA     KBCHAR
82B5 B7              ORA     A
82B6 CAB282          JZ      XX4
82B9 C9              RET

82BA CDAE82  INPUT1: CALL    GTCHA   ;Look at kbrd
82BD FE0D            CPI     0DH     ;Is is <cr> ?
82BF CA0082          JZ      CYPHER  ; Yes! End
82C2 FE20            CPI     ' '     ; -a SPACE?
82C4 CACE82          JZ      XX7     ; Y! Low # valid
82C7 00              NOP
82C8 00              NOP
82C9 FE41            CPI     65      ; N! Is it valid?
82CB DABA82          JC      INPUT1  ; No. Try again
82CE FE5B    XX7:    CPI     91      ; Yes. But really
82D0 D2BA82          JNC     INPUT1  ; valid? N! Again
82D3 CDC817          CALL    LO      ; Yes! Print it
82D6 C9              RET             ; and go back

             ;String Storage:

82D7 06020300 BXDSP: DB      6,2,3,0,10,'NORMAL TEXT:'
82DB 0A4E4F52
82DF 4D414C20
82E3 54455854
82E7 3A
82E8 03000C0B        DB      3,0,12,11,2,0,82,242,127
82EC 020052F2
82F0 7F
82F1 52FF0200        DB      82,255,2,0,72,242,127,72
82F5 48F27F48
82F9 FF              DB      255

82FA 06010300        DB      6,1,3,0,16,'ENCODED TEXT:'
82FE 10454E43
8302 4F444544
8306 20544558
830A 543A
830C 0300120B        DB      3,0,18,11,2,0,58,242,127
8310 02003AF2
8314 7F
8315 3AFF0200        DB      58,255,2,0,48,242,127,48
8319 30F27F30
831D FFEF            DB      255,239

831F 03000C06 POS:   DB      3,0,12,6,2,239
8323 02EF
```

```
                     ;If you don't have lower case, type the
                     ; strings in upper case. The mistakes
                     ; aren't mistakes. Type them in!

8325 06030C1B CLR:   DB      6,3,12,27,24,15,3,0,0
8329 180F0300
832D 00
832E 43595048        DB      'CYPHER PROGRAM: ',6,5
8332 45522050
8336 524F4752
833A 414D3A20
833E 0605
8340 546F2045        DB      'To Encode Text.',3,0,2
8344 6E636F64
8348 65205465
834C 78742E03
8350 0002
8352 06025468        DB      6,2,'This prmgram will '
8356 69732070
835A 726D6772
835E 616D2077
8362 696C6C20
8366 72656365        DB      'receive text from the '
836A 69766520
836E 74657874
8372 2066726F
8376 6D207468
837A 6520
837C 6B657962        DB      'keyboard and encode it '
8380 6F617264
8384 20616E64
8388 20656E63
838C 6F646520
8390 697420
8393 6163636F        DB      'according to the table '
8397 7264696E
839B 6720746F
839F 20746865
83A3 20746162
83A7 6C6520
83AA 6F662065        DB      'of equyvalents we have '
83AE 71757976
83B2 616C656E
```

```
83B6 74732077
83BA 65206861
83BE 766520
83C1 73657420       DB      'set up.',3,0,4,'Enter '
83C5 75702E03
83C9 0004456E
83CD 74657220
83D1 74657874       DB      'text in normil form '
83D5 20696E20
83D9 6E6F726D
83DD 696C2066
83E1 6F726D20
83E5 62656C6F       DB      'below (up to 64 '
83E9 77202875
83ED 7020746F
83F1 20363420
83F5 63686172       DB      'characters.) Coded   '
83F9 61637465
83FD 72732E29
8401 20436F64
8405 65642020
8409 20
840A 20746578       DB      ' text will be printed '
840E 74207769
8412 6C6C2062
8416 65207072
841A 696E7465
841E 6420
8420 62656C6F       DB      'below it.',239
8424 77206974
8428 2EEF

842A (001E)         DS      30      ;Fun again.

            ;Code table of equivalents:A=Z, B=Y,etc.
            ; If you change it you won't be able to
            ; read my secret messages! Only caps
            ; from A to Z and "space" are valid
            ; entries. Expand the table to include as
            ; many other characters as you like.
            ; Annex between end of table & FINISH

8448 415A4259 CODE: DB      'A','Z','B','Y','C','X'
844C 4358
844E 44574556       DB      'D','W','E','V','F','U'
8452 4655
8454 47544853       DB      'G','T','H','S','I','R'
8458 4952
845A 4A514B50       DB      'J','Q','K','P','L','O'
845E 4C4F
8460 4D4E4E4D       DB      'M','N','N','M','O','L'
8464 4F4C
8466 504B514A       DB      'P','K','Q','J','R','I'
846A 5249
846C 53485447       DB      'S','H','T','G','U','F'
8470 5546
8472 56455744       DB      'V','E','W','D','X','C'
8476 5843
8478 59425A41       DB      'Y','B','Z','A',' ',' '
847C 2020

                                    ;Note: space=space

847E (8200)         END     CYPHER  ;Don't forget (cr)
NO ERROR(S) - CYPHER.SRC;03

3 EQUATE(S)

KBCHAR  81FE    LO      17C8    OSTR    182A

13 LABEL(S)

AGAIN   826A    BXDSP   82D7    CHRINT  82A7    CLR     8325
CODE    8448    CYPHER  8200    GTCHA   82AE    INPUT1  82BA
MAIN    825E    POS     831F    PRINT   8273    XX4     82B2
XX7     82CE
```

REFRESHER COURSE

Basic will compute trigonometric functions not included in its vocabulary if you define some indentities first. The DEF function may be used in some cases, otherwise a simple statement of equality will do. (You do remember these, don't you.) The angle is given in radians. In these first identities we would set up as follows:

```
10 REM LET Y=COT(X), (Y is the cotangent of X[rad])
20 Y=1/TAN(X)
```

And here are the identities.

```
COTANGENT = 1/TAN       ARCSINE = ATN(X/SQR(1-X^2))
SECANT = 1/COSINE       ARCCOSINE = ARCTANGENT(SQR(1-X^2)/X
COSECANT = 1/SINE       ARCCOTANGENT = ARCTANGENT(1/X)
                        ARCSECANT = ARCTANGENT(SQR(X^2-1))
                        ARCCOSECANT = ARCTANGENT(1/SQR(X^2-1))
```

'ARCSINE(X)' means "..the angle whose sine is X." So the first identity below might be set up as follows:

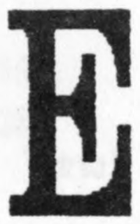
```
10 REM LET Y=ARCSINE(X) (Y is the angle[rad] whose sine is X)
20 Y=ATN(SQR(1-X^2)) : REM Arctangent(P) = ATN(P) in Basic
```

To convert degrees to radians: `X[rad] = X[deg]*PI/180`
To convert radians to degrees: `X[deg] = X[rad]*180/PI`

(Hyperbolic functions are challenging but possible. Let us know if you're hungry for them too!) □

PROGRAM LISTING:

```
0   CLEAR 100 : PLOT 12 : FLG=PEEK(33265)
10  PLOT 27,18,4,27,13 : REM OUTPUT TO RS232 PORT
12  PLOT 27,65,8
15  FOR L=1 TO 7
25  PLOT 27,76,48,0 : REM FOR SINGLE DENSITY USE 75 INSTEAD
26  REM                      OF 76. ALSO GIVES WIDER LETTER.
30  FOR N=1 TO 48 : READ R : PLOT R : NEXT N
70  PLOT 13,10 : NEXT L

80  DATA 3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
85  DATA 3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
90  DATA 224,224,224,224,224,224,240,248
92  DATA 255,255,255,255,255,255,255,255,255,255
95  DATA 224,224,224,224,224,224,224,224,224
96  DATA 224,224,224,224,224,224,224,224,224

100 DATA 240,240,248,252,254,255,255,255,255,255,255,255
105 DATA 0,0,0,0,0,0,0,0
106 DATA 255,255,255,255,255,255,255,255,255,255
110 DATA 0,0,0,0,0,0,0,0,0,0
112 DATA 3,127,127,127,127,127,0,0,0,0,0,0,0,0
114 DATA 128,240,255,255,255,255,255

115 DATA 0,0,0,0,0,0,0,0
116 DATA 255,255,255,255,255,255,255,255,255,255
117 DATA 31,31,31,31,31,31,31
120 DATA 63,127,255,255,255,255,255,255,255
121 DATA 0,0,0,0,0,0,0,0,0,128,128,128,128,128

125 DATA 0,0,0,0,0,0,0,0
126 DATA 255,255,255,255,255,255,255,255,255,255
127 DATA 0,0,0,0,0,0,0,128,192,224,248
128 DATA 255,255,255,255,255

130 DATA 0,0,0,0,0,0,0,0,0,63,63,63,63,63
135 DATA 0,0,0,0,0,0,0,0
136 DATA 255,255,255,255,255,255,255,255,255,255
137 DATA 0,0,0,0,0,0,0,0,0,0,0
138 DATA 224,224,224,224,224,0,0,0,0,0,0

140 DATA 1,7,63,255,255,255,255,255
145 DATA 31,31,31,31,31,31,63,127
146 DATA 255,255,255,255,255,255,255,255,255,255
150 DATA 31,31,31,31,31,31,31,31,31,31,31,31,31
151 DATA 31,31,63,63,63,127,127,127
155 DATA 255,255,255,255,255,255,255,255,255

160 POKE 33265,FLG : END
```



—double density printing
(see line 25 of program listing: PLOT 27, 76, 48,0

FIG 1.

—single density printing
(see line 25 of program listing: PLOT 27,75,48,0

Block Number	Value
7	0
6	0
5	0
4	0
3	0
2	0
1	2
0	1

FIG 3.

Total Value = 3

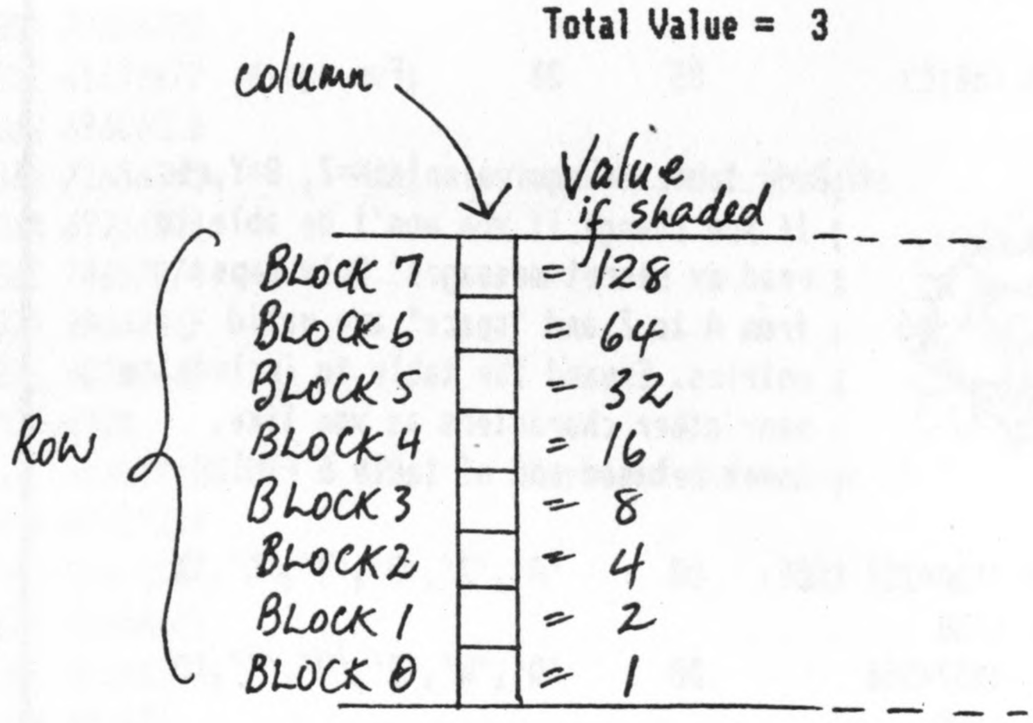


FIG 2.

Ric Lowe
80 Cawston Road

Attadale 6156
WEST AUSTRALIA

Many Compucolor users who own an Epson printer (Type II or III) may have wished that they could use the bit image mode, but have given up when faced with the lack of specific information in the Epson operating manual. This article will show you how to develop a program to print a large upper case letter 'E' as depicted in FIG 1. The same general technique can be used to develope code for any user-generated symbol, such as music symbols, logo's, line drawings, etc.

The following steps give a general method for translating the desired shape into the numbers that will cause the printer to create that symbol in print.

STEP 1. Obtain a suitable original for the symbol you wish to 'draw' with the printer (in our case, the letter 'E').

STEP 2. Prepare a sheet of transluscent graph paper by ruling lines across it at intervals of every 8 divisions. Paper with 1 millimeter divisions is useful for this purpose.

STEP 3. Trace the original onto the graph paper after positioning the original so it is aligned with the divisions on the graph paper.

STEP 4. Calculate the bit image codes that represent the symbol. These codes are the numbers that tell the printer how to generate the symbol. They are calculated by considering each of the 8 division intervals you produced when lines were ruled across the graph paper in STEP 2. The effect of these lines is to carve the symbol into horizontal 'slices', with each slice being 8 divisions high.

We can pause here to establish some terminology before we begin these calculations.

A. A "row" will be the term we use to describe the slices produced by ruling lines at each 8 division interval. In the case of the letter 'E' we have 7 rows; Row 1 makes up the top of the letter and Row 7 makes up the base of the letter.

B. Each row is made up of a series of adjacent vertical "columns" that run from the top row to the bottom row. These columns are numbered from left to right. Our letter 'E' extends across the graph paper for 48 divisions, so we number our columns 1 to 48.

C. Each column is made up of 8 "blocks" which we will identify as Block 0, Block 1, Block 2, ...Block 7. Each of these blocks is an individual square on the graph paper.

Each block within a column has a particular value assigned to it that is used if we want that block to be printed. These values are shown in FIG 2. If we don't want a particular block printed, we give it the value '0'. An example will make this clear. Let's consider the very left hand corner of our letter 'E'. (See FIG 4.)

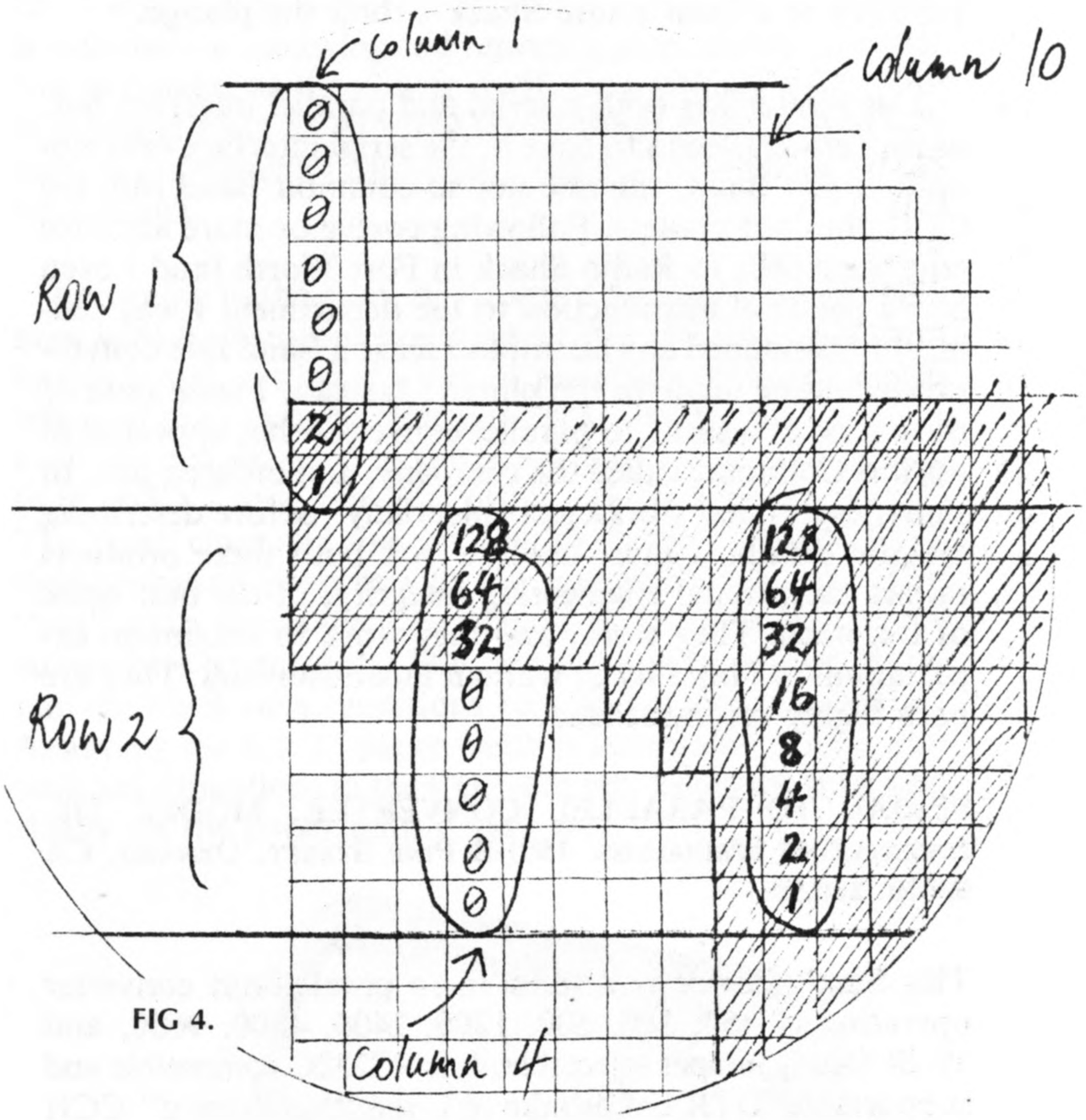


FIG 4.

We identify this as Row 1, Column 1. By looking at the blocks within this column we can see that only Block 0 and Block 1 are shaded in. This gives us the values in FIG 3.

This means that the number '3' is what the printer needs as its bit image code for printing this upper corner of our letter 'E'.

Now have a look at Row 2 and see if you can work out the reason for a bit image code of '224' for the 4th column and a code of '255' for the 10th column.

STEP 5. When all of the columns in all the rows used to represent the letter have been converted to numbers as in STEP 4, plug these numbers into the data statements as shown in the program listing. For example, you can see that program lines 80 and 85 contain the 48 numbers needed to represent the top of the letter 'E'. Program lines 90, 95, and 100 contain the 48 numbers needed to represent the next section of the 'E' and so on.

NOTE: For symbols that require a different number of rows and columns than those used in our example, the value of 'L' (rows) and 'N' (columns) must be changed accordingly.
□

A Serial-to-Parallel Converter

PRODUCT REVIEWS: Radio Shack CGP-115 Pen Plotter

Joseph Norris

These two products are reviewed together because I use them together. I have long wanted to play with a pen plotter and have not felt comfortable with the $1000 price tag on the "table models" available. When I saw the CGP-115 on sale for $119 at a local Radio Shack I took the plunge.

This plotter has both a serial and parallel interface but, as luck always seems to have it, the serial interface operates only at 600 Baud, the one not-so-common Baud rate the CCII does not possess. Following twelve or more abortive telephone calls to Radio Shack in Fort Worth (and I even had a personal introduction to the department I was calling!) I abandoned any attempt to see if a Baud rate conversion could be made in the plotter. Instead, I took note of the notice of a serial to parallel converter that appeared in the last Colorcue, called the company, and ordered one. In short, everything works just splendidly. Before describing specific details, I must observe that both these products reflect exceptional engineering integrity, in the best sense of the word. They work, and they work to maximum expectations, without bugs, without inconsistancies. They are both highly recommended.

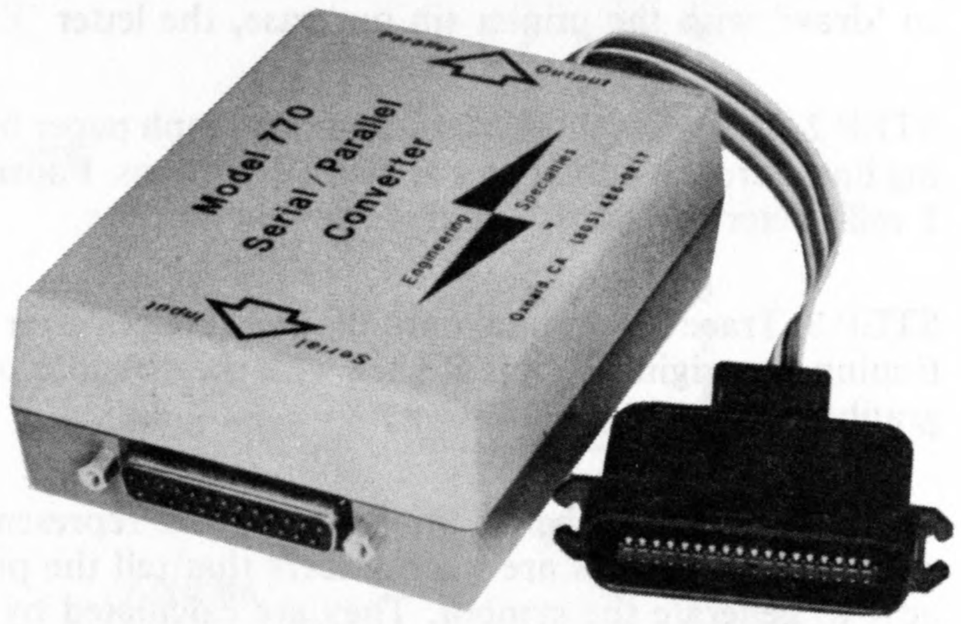
SERIAL TO PARALLEL CONVERTER, MODEL 770. ENGINEERING SPECIALTIES. 1501-B PINE STREET, OXNARD, CA 93030. $89.95.

This handy device is a serial-in to parallel-out converter operating at 150, 300, 600, 1200, 2400, 4800, 9600, and 19.2K Baud, jumper selectable. It is RS232C compatible and supports the DTR handshake (e.i. the "handshake" CCII modification). The input connector is the DB25S, the female version. It operates with no parity, 8-bit format, and 1 or 2 stop bits. The parallel output uses BUSY and STROBE, which may be polarity inverted at the user's discretion. The 8-bit port is Centronics compatible and has latched outputs. The implied parallel connector is an AMP 552470-1 or its equivalent. I say "implied" because the unit will come without an output cable and connector unless it is ordered with this option for a $10 additional cost. I ordered this option because it is often very inconvenient to get the parts together.

The converter is very well made, using good parts, a professional circuit card, and a design that reflects the intent to do one's best. It is housed in a two-piece plastic box, and receives its power from either the host computer or the peripheral. A separate outboard power supply (not supplied) may also be used, but should not be required in most cases. The converter draws about 20 milliampheres at 5 volts D.C. I used the pen plotter to supply this power through pin 18 of the parallel port connector. An optional provision is made for taking power from any selected pin of the DB25S input connector by means of a moveable pin. Complete instructions are provided in the accompanying manual, 8 pages long, which leaves no stone unturned.

I'll admit my feelings about this converter are colored by a very pleasant experience with the factory, whose personnel are unpretentiously courteous, concerned that their product work well in my application, and, by golly, who know how to listen when you speak to them.

Only good things happen when you acquire a serial-to-parallel interface. Parallel port printers are less expensive that serial port models, and this little box will be all the interface you will need, short of buffer storage. There are other devices, useful with the CCII, or any serial output computer, that have only parallel interfaces. (The parallel interface is less expensive to make and thus is the interface of choice for peripheral equipment.) Much of the more exotic peripheral hardware you may see on the market may be used with the CCII and this interface box. It is a "freeing" device to have. The price is right, and I assure you the factory support is unequalled in my experience.



PEN PLOTTER,MODEL CGP-115. RADIO SHACK. $119-195, DEPENDING ON SOURCE.

A loveable little box about the same weight as a large loaf of good homemade bread, the CGP-115 is an engineering marvel. It is a four-pen plotter supplied with "ball-point" pens in red, green, blue, and black. The paper is ordinary calculator paper, 4-1/2 inches wide, roll feed. Three external controls operate Power, Paper Feed, and manual Color Selection. A bevy of software commands cover every conceivable manipulation of the plotter.

The instruction manual is adequate but leaves a few things unsaid, such as the necessity for carriage returns after some of the commands—as opposed to entering multiple statements on one line when using Basic. But you will do well as it is, with very few frustrating moments. (They can't take that away from us, can they'...the frustrating moments.) A few demonstration programs are included, like those that made some of the illustrations on the cover of this issue.

I programmed the map of the United States in about three hours (a tour de force, my friends!) and it took about 15 seconds, on each pass, to print. If you look at the (unretouched) printout through a lens, observe that I made four passes of this particular plot to intensify it for reproduction and that the overplot error is less than 0.001''. In my early electronic career, I was engaged in the design of sophisticated servo-mechanisms for navigational computing and I can tell you this mechanism is superb and works without flaw. To watch this plotter in action is simply awesome.

What will it do? Well, anything you have the patience to program. A summary of available commands will best tell the story, but I imagine anything you can draw with pen and paper, the CGP-115 can draw too. A text command mode and graphics command mode control plot head and paper movement. Text mode prints text only and responds only to the CTRL commands below. Text is printed in nearly-continuous point sizes from 4 points to 112 points, selected by the user (that's about 2 millimeters to over seveal inches high). Text may be rotated to print vertically up or down, and even upside down. The graphics mode also supports text, as well as line and curve drawing. Here are the commands and note that they may be used in assembly language routines as well as any other language that supports ASCII transmission to the output port.

CONTROL CODES:

CTRL 0 = backspace; 10 = linefeed; 11 = reverse linefeed; 13 = carriage return; 17 = select text mode; 18 = select graphics mode; 29 = rotate pen holder (color select).

GRAPHICS COMMANDS:

[A] Return to text mode.
[Cn] Change to specified color n = 0 to 3.
[Dab,cd,ef....] Draw from specified xy coordinates a and b, relative to currently established origin, to cd, to ef ...
[H] Home the pen to currently specified pen origin.
[I] Set pen origin to current pen position.
[Jab,cd,ef....] Draw line from current pen position to relative increments ab, then to cd, then to ef....
[Mxy] Move pen without printing to absolute coordinates xy, relative to current pen head origin.
[Ln] Set to line type n = 0 to 15. Sixteen different line types from solid to varying dash configurations.
[P] Print following quoted ASCII string as text.[Sn] Select printing character size n = 0 to 63 units high.[Qn] Rotate to print direction n = 0 to 3, 90 degrees apart.[Rxy] Move without printing to coordinates xy relative to current pen location.
[Xabc] Draw an axis line, where a = 0 is x-axis, a = 1 is y-axis, b = distance between gradation marks, and c = number of marks to be drawn.

This last command, executed twice, draws a complete axis format at lightening speed. I have used it to draw a four-sided axis-lined ''box'' about 3 inches square, in less than five seconds. If you are familiar with the commands of other plotters, you will notice that there is no 'pen up' or 'pen down' command. The CGP-115 uses the [M] and [R] command modes instead. Other differences are observable as well, but are not indicative of performance limitations.

The CGP-115 has four DIP switches that select 40 or 80 characters per line, serial or parallel input, carriage return or carriage return/line feed on CTRL 10, and 7-bit or 8-bit ASCII recognition. There is a small internal character buffer, I would guess about 8 bytes deep. A very clever facility is included for removing and replacing pens, almost as much fun to operate as the plotter itself. Low voltage AC power is supplied from an external transformer (included) which may be permanently connected. The power switch is on the back panel.

Returning to the map, it was made by overlaying a 'real' map with transluscent graph paper (see Ric Lowe's article in this issue), setting the pen to an origin (I) at the upper left, and then moving relative to that origin (Jxy,xy,xy...) until the pen returned to the point from which it began. A FOR..NEXT loop moved it around the course four times. The ''pie'' chart and sine curve software were adapted from demonstration programs in the operator's manual. (The ''pie'' chart shows the geographical distribution of current subscribers by number.) They are actually all in four colors and the black-only renditions here do not do them justice. Assuming the 4-1/2″ paper width is acceptable, I have not seen any limitations in the CGP-115 yet. You may draw vertically on the paper until it runs out.

I have used this plotter to print a survey of weekly sales for my company. The inital response was ''Really, Joe. Well that's interesting, (ho-hum)'', but in fact if I'm an hour late on Fridays showing my ho-hum graphs for the current week, there is a parade of 'disinterested' spectators at my office door, waiting to see the new print. I draw bar graphs of data, which are very helpful when I review experimental progress history. It saves a tedious rereading of notes when I have been away for awhile and want a quick refresher course on where things stand and how they got there. This is, in truth, a money and time saving application. The most valid application, from my perspective, has been having more fun than a barrel of monkeys. If it happens that your lust for this kind of power overcomes the more sensible dictates of your wallet, be sure to buy an extra set of pens and another roll of paper. You'll want them in short order. □

BASIC'S FILE STRUCTURE: A Review

Opening the File: The FILE "R" Statement.

A random file must be 'opened' before its data is available to a BASIC program. The BASIC file open command string has this general form .

FILE "R",C,F$,BUF;R,N,B

where F$ is the file name, R is the number of records, N is the number of bytes in each record and B is the number of records that will be brought into each buffer in computer memory at one time.[1]

The two new parameters are C and BUF, both to be defined in a moment. The FILE "R" command assumes that the specified file exists on the disk being accessed, and will return an FCS error message if the file is 'not found.' The FILE 'R' routine parses the file specification for accuracy and checks the disk directory to see if the file exists. It creates buffer space and sets up pointers to the disk file, keeping track of reads and writes.

The parameters of the FILE "R" statement will be very like those used to create the file. The product of the parameters R and N (R*N) in the FILE "R" statement must equal the product of the like parameters that were used when the file was created with the FILE "N" statement. We do not say R and N must be the same in both cases, only that their product must be the same. We have seen that the product R*N defines the space on disk in which the file resides, and this space determination must remain a constant, for any given file, in all file commands.

The values given to R and N in the 'open' command string are usually assigned in such a way that any record (R) will hold equivalent record fields of the same length and in the same order. Logically, one would determine the record length first (N), and then indicate the total number of records on file (R), then decide how many records must be accessed in one disk read (B).

The parameter C is a number used to refer to an opened file within the context of the BASIC program, and is a decimal number between 1 and 127. It is used to place a temporary shorthand label on the particular file being opened, for use by the computer. If TEST.RND is opened first with C=1, the computer will call it File 1. This saves memory space because '1' is shorter to save in a memory directory of opened files that 'TEST.' A second file to be opened in addition to File 1, and at the same time, may be opened with C=2, and a third with C=3 and so on. BASIC does not require that sequential values of C be employed, nor that the first file have C=1. C is a label for the file in memory,that allows the computer to distinguish among opened files, and has no governing regulations except that the value of C may not exceed 127. The first file opened may have C=6, for instance. If only one file will be open at any given time, all files may have C=1.[2]

The parameter BUF is a decimal number between 1 and 255 which determines the number of file buffers that will be established in user space by BASIC. This buffer is a storage area into which the file contents is read, and is located in the array space area assigned to the BASIC program. If there is no shortage of user space, only one buffer will be required, assuming that the time required to read an entire disk file into that buffer is not unsatisfactory.

File buffer space is allocated in memory in 128-byte increments, and no file buffer will be smaller than 128 bytes,even if all that space is not required by the FILE "R" parameters. For example, SAMPLE.RND with 2 records of 64 bytes will have 128 bytes allocated for its storage in memory. SAMPLX.RND with 1 record of 257 bytes will have three 128-byte areas allocated for its memory storage. The reserved user memory space is actually increased, for each opened file, by the 34 bytes required for the File Parameter Block for that file, 12 bytes for the computer's file scratchpad per file,and 4 bytes for record keeping on each buffer allocated to that file. In the following discussion, only the actual requirement for file data space will be meant when space allocations are specified.

A single file buffer may have, in theory, 32767 * 32767 * 255 bytes (— the maximum values for R, N, and B = 2.737 * E11 bytes, which could put most computers into convulsions). The realities of available memory space demand a more considered allocation of buffer space, however. Multiple file buffers are usually employed when user space is in short supply and file record access requires holding specified records for quick access.

Assume a file EXAMPL.RND, 16,64,B is to be opened with one buffer (BUF = 1). It is permissable for B to take on values between 2 and 16. (It may not have a value of 1 because file access is made in integer multiples of 128 byte diskblocks, 2*64.) If we were to set B = 16 and open this file, only one disk

```
10 REM TEST.BAS, following the FILE "R" statement operation.
12 REM           First, create FILE "N","TEST.RND",16,128,16
16 GOTO 200
20 REM           'PROGRAM' DISPLAY SUBROUTINE

24 PLOT 3,2,7  : PRINT " 68 OPEN FILE"
26 PLOT 3,2,9  : PRINT " 74 GET 1,2;A$[24]"
28 PLOT 3,2,11 : PRINT " 80 GET 1,3;B$[24]"
30 PLOT 3,2,13 : PRINT " 86 PUT 1,2;B$[24]"
32 PLOT 3,2,15 : PRINT " 92 PUT 1,3;A$[24]"
34 PLOT 3,2,17 : PRINT "102 CLOSE FILE"
36 RETURN
38 REM           AUXILIARY SUBROUTINES

42 PLOT 3,42,Y : PRINT "FREE MEMORY = ;FRE(X) : RETURN
44 PLOT 6,0,3,65,30 : INPUT "";Z : PLOT 6,2 : RETURN

48 REM           MAIN PROGRAM

52 PRINT "FILE 'R',1,'TEST.RND',<buf>;16,128,<b>" : GOSUB 42
54 PLOT 3,42,2 : PRINT "HIT <RET> TO EXECUTE" : PLOT 3,42,3
56 PRINT "STARRED PROGRAM LINE..." : GOSUB 24 : PLOT 3,0,2
58 INPUT "ENTER BLOCKING FACTOR (B) > ";B
60 INPUT "ENTER NUMBER OF BUFFERS (BUF) > ";BUF
 64 PLOT 3,0,7 : PRINT "*" : GOSUB 44 : PLOT 3,6,7
66 PRINT "FILE 'R',1,'TEST.RND',";BUF;";16,128,";B
68 FILE "R",1,"TEST.RND",BUF;16,128,B : Y=7 : GOSUB 42

72 PLOT 3,0,9 : PRINT "*" : GOSUB 44
74 GET 1,2;A$[24] : Y=9 : GOSUB 42 : PLOT 3,0,9 : PRINT " "

78 PLOT 3,0,11 : PRINT "*" : GOSUB 44
80 GET 1,3;B$[24] : Y=11 : GOSUB 42 : PLOT 3,0,11 : PRINT " "

84 PLOT 3,0,13 : PRINT "*" : GOSUB 44
86 PUT 1,2;B$[24] : Y=13 : GOSUB 42 : PLOT 3,0,13 : PRINT " "

90 PLOT 3,0,15 : PRINT "*" : GOSUB 44
92 PUT 1,3;A$[24] : Y=15 : GOSUB 42 : PLOT 3,0,15 : PRINT " "

 96 PLOT 3,0,20 : PRINT "HAVE YOU NOTICED THE DISK ACCESSES?"
 98 PLOT 3,0,22 : INPUT "<Q> TO END, <R> TO RECYCLE > ";Z$
100 PLOT 3,0,20,11,3,0,22,11,3,0,17 : PRINT "*" : GOSUB 44
102 PLOT 3,0,17,11,3,2,17 : PRINT "102 FILE 'C',1"
104 FILE "C",1 : Y=17 : GOSUB 42
106 PLOT 3,0,21 : PRINT "CAN YOU TELL WHY FREE MEMORY IS ";
108 PRINT "LESS AT THE END OF THE PROGRAM?"
110 PLOT 3,3,23 : PRINT "NOTICE THAT THE ACCESS TIME IN ";
112 PRINT "LINE 92 IS LONGER BECAUSE"
114 PLOT 3,7,24 : PRINT "A MODIFIED RECORD MUST FIRST BE ";
116 PRINT "WRITTEN TO DISK."
118 IF Z$<>"R" THEN PLOT 3,0,27 : END
120 PLOT 3,0,27 : INPUT "PRESS <RET> TO CONTINUE";Z
122 RUN
200 CLEAR 1000 : PLOT 6,2,12,27,24,15 : GOTO 52
```

access would be required for file data and all records would be simultaneously available for access with GET and PUT statements.

If, however, B = 2, then records 1 and 2 only would be read into the buffer. We can use GET and PUT with these two records without additional access. To GET record 3, the contents of the buffer would be overwritten by records 3 and 4. Should we now wish to GET 2 again, for example, the disk would be accessed and the buffer overwritten with records 1 and 2 again. It would be more convenient to set B = 4 and read all four records into a 256-byte buffer at once.

Now consider NEXTFL.RND, 16,256,2. The file buffer will hold 512 bytes, or two file records, at a time. The operations with this file, in one file buffer, will be the same as described above. Suppose that I want to exchange the record data between records 2 and 3 as follows:

```
150 GET 1,2;A$[24] : REM Records 1
& 2 written to buffer.
155 GET 1,3;B$[24] : REM Records 3
& 4 written to buffer.
160 PUT 1,2,B$[24] : REM Records 1
& 2 rewritten to buffer.
170 PUT 1,3;A$[24] : REM Records 3
& 4 rewritten to buffer.
```

Four disk reads are required to perform this operation with the one allocated buffer. If BUF = 2, then two 512-byte buffers will be allocated. When line 150 is executed, Records 1 and 2 will be read into Buffer 1. When line 155 is executed, Records 3 and 4 will be read into Buffer 2. Lines 160 and 170 will not require any further accesses.

Should we now GET 1,7;A$[24], one of the buffers will be overwritten with Records 7 and 8, specifically, the buffer that was not used last. Since Record 3 was accessed last, in line 170, the buffer holding Records 3 and 4 would remain, and the buffer holding Records 1 and 2 would be overwritten with Records 7 and 8. This "leapfrog" procedure will continue, preserving the last-accessed buffer and overwriting the other. If many buffers are used, the one whose contents were most distantly accessed or altered will be overwritten.

If the contents of any buffer have been altered, the buffer contents will be rewritten to disk before new records are read into that buffer. This is equivalent to an automatic FILE "D" operation. This is not true for PUTs to the last buffer written by the disk. Only a FILE "C" command will rewrite changes in this last-to-be-written buffer.

The value assigned to BUF interacts with the blocking factor parameter (B) to determine which records of a file maybe read from user memory without further disk access. In the case of NEXTFL.RND (above) a FILE "R" specification of 16,256,4 and one buffer is preferable to a specification of 16,256,2 and two buffers, because it requires one less disk read and slightly less memory space. In the case where Records 1 and 16 will be exchanging data, much less user memory is required if BUF = 2, and Records 1 and 2 are in one buffer, and Records 15 and 16 are in the other. Using two buffers prevents the allocation of user memory for all 16 records. It is clear why the authors of the CCII Operator's Manual did not concern themselves with the buffer allocation. Unless the 'leap-frog'' procedure, just described, is intricately woven into the fabric of a BASIC program, additional buffers serve no useful purpose.

Observe that the FILE "R" statement does not read data into the buffer. The disk activity one hears is the checking of the disk directory for accurate file parameters. It is the GET or PUT function that actually "triggers" the file data read. The following BASIC Listing will demonstrate the procedures discussed here in a simple way. Disk accesses can be interpreted by listening for the drive motion, and free memory space at each stage of the procedure will be displayed on the screen. This listing is readily modified to test other Disk BASIC operations. □

[1] See "Basic's File Structure" in Colorcue, Vol VI, N0. 1 for descriptions and restrictions on the F$, R, N, and B parameters.

[2] Similarly, files may be closed by number in any order or in any combination of orders without restriction.

Compu-Free Colorware

Take the risk out of buying software. Compu-Free Colorware is user supported software. Send your disk in a mailer, and include return postage (the same amount it takes to send the disk), and I will return the disk with the program of your choice copied onto both sides. Alternately I can copy two separate programs, one on each side. Each program includes a manual on the disk in a .SRC file along with a simple minded Basic program to print it out. Use the program for several days. Try your data in it, see if it will do anything useful for you. If not erase the disk and all you are out is a couple bucks for the postage. If you like what you see, then we ask you to send a donation directly to the author of the program. The request will be spelled out in the Copyright message, along with the authors name and address. You are also encouraged to copy and share the program with your user group, and Compucolor friends. They would be under the same honor code to try the program and pay if they like it, or erase it if they don't.

Here is our current library:

Program Name:	Author:	Description:
Wordy	Dinsmore	Word Processor with dictionary. (Debugging now, ready about June 84)
Book	Dinsmore	Programmed textbook teaching aid.
Intro. to Assembly	Dinsmore	Text for Book
Assebly Language	Dinsmore	Text for Book
Assembly Applicaton	Dinsmore	Text for Book
Pottery Making	Dinsmore	Text for Book
Survey	Dinsmore	Conduct your own opinion survey.
Stock Market	Dinsmore	Tracks price of stocks from newspaper
Budget	Dinsmore	Will handle a home budget.(32K)

Programmers: Let me register and distribute your programs through the Compu-Free Colorware system. You receive payment for your programs directly from the users. I handle duplicating and advertising. Send a SASE for complete instructions for submitting your work, and the standards to which your program and documentation should measure up to.

ORDER FORM: COMPU-FREE COLORWARE

Gary Dinsmore's
Creative Software
32695 Daisy Lane
Warren Or. 97Ø53

Program name: _____ (one program copied both sides.
Second name: _____ (backside of disk)

Name: _____ [ ](place me on mailing list)
Address: _____
City: _____, State _____ Zip _____
Send a disk for each request. Send the return postage with request.

(ADVERTISEMENT)

A Pascal for the Compucolor II: Part I.

Doug Van Putte
18 Cross Bow Drive
Rochester, NY 14624

A subset of standard PASCAL, called "Tiny-Pascal", is available to the Compucolor user, and potentially to the Intecolor 3650 series user. Its primary value is that of a Pascal language teaching tool for those who are interested in an introduction to this remarkable language. At the present time, "Tiny-PASCAL" lacks some key features, such as file access, real arithmetic, and plotting capability, so its utility as a general programming language is limited. However, since it does contain the key structure and commands of Pascal, it can be used effectively to learn the language.

In a series of articles, I plan to introduce the reader to "Tiny-PASCAL", proceeding in a step-by-step fashion. This first installment will attempt to present some argument for learning Pascal, and provide information on obtaining and installing the necessary software. In the next articles I will present the language structure, the Tiny-Pascal commands, and examples of programs to illustrate their application.

Before proceeding, it must be explained that the version of Tiny-Pascal available for the Compucolor is written, interestingly enough, in the FORTH language, and uses many facilities of FORTH to execute the editing, compiling, and disk handling required by Tiny-Pascal. For this reason, it is not possible to discuss Tiny-Pascal without introducing some properties of FORTH at the same time. In fact it will be difficult, particularly in these early paragraphs, to separate the two. Nevertheless, it will all become clear in good time. You may find yourself puzzled by the use of the word "screens" in the text. This word is used in FORTH to designate a screen display, containing a maximum of 1028 bytes of program data, which constitutes a program or a portion of a program in FORTH. The contents of these screen displays are stored on disk, in 1028 byte sections, just as SRC and PRG files are stored. A "screen" is really the SRC code for a FORTH program. Any uncertainty you may feel about the concept of a "screen" will vanish when you have seen one and used it. Just keep in mind that when we speak of a "screen" we are speaking about a "screen's-worth" of program code.

Pascal was created and introduced by Niklaus Wirth in the early 1970's. Wirth named it after the French mathematician, Blaise Pascal, rather than follow the customary use of an acronym. Originally introduced as a teaching tool for programmers, Pascal has become a successful commercial language for large programs. Its importance as a teaching language has not been diminished by this success; indeed, Pascal has recently been designated as the single required language for advanced placement courses in computer science for high school students. I suspect this is because the principles of structured programming can be taught so effectively through the use of Pascal. The rigid principles demanded by Pascal's control structure help the programmer to write better programs in any language.

What does structured programming have to do with "good" programs? Quite simply, Pascal's control structures (i.e. program design) facilitate the development of programs that are likely to run correctly the first time; if not, they will be easy to modify. A Pascal program is made of relatively small "modules", or program sections, each of which is a structured program. This modular approach, by which one breaks a problem into small easily-designed segments, can hasten program definitions and facilitate the production of appropriate code. Names can be assigned to these modules and to the variables they contain, which makes the program easy to read and understand. When errors occur, their location can quickly be pinpointed and corrected, usually within a single module.

Tiny-Pascal was implemented by T. J. Zimmer in 1981, using the fig-FORTH language. This version was derived from a series of three articles published in BYTE in 1978 by K. Chung and H. Yuen, which introduced a Tiny-Pascal compiler written in BASIC. This compiler was converted to ISC Basic by T. G. Price in 1980, and resides, without documentation, on FORUM (CCUG) disk No. 13B. Zimmer's version of Tiny-Pascal was installed on the CCII in 1983 by Dr. James Minor.

The FORTH and Tiny-Pascal implementations are available from the CHIP Compucolor/Intecolor Users Group of Rochester, NY. The FORTH disks are required to support the Tiny-Pascal compiler. To obtain both the FORTH and Tiny-Pascal disks, one must be, or become, a member of the CHIP group. Annual membership is $10, and may be sent to Gene Bailey, 28 Dogwood Glen, Rochester, NY 14625. For the disks themselves, we prefer to receive two blank disks, pre-formatted (both sides), and $3.00 for shipping and handling. If CHIP supplies the disks, a $10 fee is charged. Manuals will be supplied by the CHIP library, if requested, and must be returned within 30 days; they may be purchased for $15.00. Be sure that the software version is specified with the order (v6.78, v8.79, v9.80. CHIP is in the process of finding a librarian for 8000 Series Intecolor computers also.)

Both the FORTH and Tiny-Pascal implementations are supported with excellant documentation by Jim Minor (see references 1 and 2.) The manuals contain step-by-step instructions for installing both the FORTH and Tiny-Pascal compilers. They also contain documentation to support the languages and editors, and they contain copies of the compiler source screens (for Pascal only) and of sample program screens. Please note that neither manual is a tutorial. (For detailed programming assistance on FORTH, Jim recommends reference 3; I recommend reference 4 as a Pascal programming resource.)

Once the disks are on hand, one need only follow the simple, instructions in reference 2 to configure the FORTH system for Tiny-Pascal. Those new to FORTH will refer to reference 1 for its installation. (In brief, the FORTH compiler is loaded through FCS, and used to compile the Tiny-Pascal compiler. The FORTH compiler, now augmented with Tiny-Pascal, is re-saved as an extended version of FORTH. This new version is called TPAL4, by Jim Minor, and can be run from FCS to compile or to "run" any of the FORTH or Tiny-Pascal programs.

The user can now add an editor to TPAL4. The manual, again, presents easy-to-follow instructions to do this. The editor will be used to enter programs similar to the way SRC code is entered for an assembly language program. A line editor is provided with the Tiny-Pascal sample "screens" (which is superior to an alternate editor supplied with the FORTH sample screens). The Tiny-Pascal editor is located on screens 20 through 37 on the Tiny-Pascal sample screen disk. The procedure is to first install the FORTH line editor as described in reference 2, Appendix 6. (I label it TPALED.) Now refer to the instructions in reference 1, page 10, to augment TPAL4 with the line editor. Replace the "6 LOAD" command with a "20 LOAD" command. (Don't forget to load this from the 'starter screen' disk in the Tiny-Pascal set, rather than from the FORTH set.)

Some simple FORTH commands will have to be learned in order to use the line editor. Here is a description of these commands (each command is terminated with the traditional "carriage return.")

[NUM] LOAD;Compile program beginning on screen [num]. Note that the screen number is entered first. Always observe the spaces shown in these commands.

[NAME] ;Run the compiled program called [name].

SAVE [NAME] ;Save the compiled program to disk with the Tiny-Pascal compiler. (This procedure is not normally recommended for reasons to be described later.)

FORTH FORGET [NAME] ;Erase the compiled program in memory called [name]. This does not erase programs on the disk.

EDITOR ;Invokes the augmented version of the editor.

PASCAL ;Invokes the Tiny-Pascal compiler (before compiling a program.)

FLUSH ;Writes all changes made on the screen editor to disk. This procedure is recommended upon completion of screen editing (before going on to another screen).

1 PRINTF ! ;Sets a flag to print to the serial port and to the CRT.

0 PRINTF ! ;Resets the above flag to print to the CRT only.

[N] 3PRINT ;Prints 3 screens beginning with screen [n]. This is a compile command from screen -19 of the FORTH Starter Screens.

FCS DIR ;Print the disk directory to the device(s) shown by the print flag. (See above.)

FCS [COMMAND] ;Execute any FCS command.

To set the printer Baud rate from FORTH, type [CPU RESET], [ESC] [R] followed by the Baud Table number (7 = 9600 Baud, etc.) To return to FORTH, type [ESC] [T].

With these fundamental commands in hand, we can look at the FORTH, Inc. Line Editor. The editor, recommended in reference 2, Appendix 6, is a "line" editor. Those familiar with "screen" editors will find any line editor rather clumsy to use. However, it's the best we have, and surely better than the editor in reference 1 in terms of its flexability and ease of use. The editor permits one to enter, delete, and edit text sufficiently to prepare programs for either the FORTH or Tiny-Pascal compiler. Unlike BASIC, where one can enter and immediately "interpret" (run) a program, Tiny-Pascal and FORTH require an additional step. Following the typing of the program code onto the screen, the program must be compiled, using the commands given above. Any errors indicated during compilation will have to be located and corrected. Following an error-free compilation, the compiled program may be "run" simply by typing in the program name. If it is desired to "save" the compiled program in a PRG version of Tiny-Pascal, the SAVE command will be used (see previous command list). This SAVE adds to the Tiny-Pascal compiler, which could soon become quite large with programs, and therefore, inefficient.

The line editor commands are, for the most part, single letter commands. Some are followed only by a space, others by text to be used with the command. There are three buffers in memory used by the editor to store the text portion of your commands. If a command requiring text is entered without the text, the editor will supply a "default" text, taken from the last text buffer to have been used. This facility may be used to place the same text on multiple command lines.

The following is a list of editor commands divided into three groups; a)screen utility commands, b)line-type commands, and c) basic editing commands.

SCREEN UTILITY COMMANDS:

L ;List the current screen.

[NUM] L ;Copy screen [num] from disk to memory and make it the "current screen."

N ;Make the "current screen" the next screen, [num] + 1.

B ;Change the "current screen" to the previous screen, [num]-1.

COPY [NUM1] [NUM2] ;Copy to disk, from Screen 1 through Screen 2.

WIPE ;Clear the current screen in memory.

LINE-TYPE COMMANDS:

[NUM] T ;Position the cursor at line [num], also called the "current line", and type the line.

X ;Remove the "current line" and move all lines below it up one line.

U [TEXT] ;Insert this text line below the "current line". Push all other lines following down one line.

TILL [TEXT] ;Delete the contents of the "current line" from the cursor up to and including [text].

P [TEXT] ;Replace the contents of the "current" line with [text].

[NUM1] T X [NUM2] T P ;Type line [num1], remove it, move all lines following up

one line; type line [num2], put line which was line [num1] where line [num2] is, and erase line [num2].

BASIC EDITING COMMANDS:

I [TEXT] ;Insert [text] in front of the current cursor location.

D [TEXT] ;Delete [text] in the "current" line.

F [TEXT] ;Find [text] in the "current" line.

F [TEXT1]^ R [text2] ;Find [text1] and replace it with [text2]. The [^] (lower case "USER" key) terminates a text string to permit another command to follow.

As can be seen from some of these examples, multiple commands may be entered in the same line. To do this successfully, any [text] entry must be terminated with the USER carot [^] and spaces must be inserted between command strings. In addition to the commands above, there are several "search" and "move" commands available. You will find these in reference 2, Appendix 6.

Well, now that we've seen to the installation of Tiny-Pascal, let's look to our future in using it as a programming language. As an example of things to come, I am using the listing that follows to show what a Tiny-Pascal program looks like. The program actually begins at the first "BEGIN" statement, and ends at the last "END" statement. The punctuation included here is absolutely essential, but the spacings are entered to make the program more readable. You will note some similarities to words in BASIC.

This program prompts the entry of ten integers, sorts them in ascending numerical order, and prints them out in ordered pairs. This program introduces the concepts of declaring constants, variable identifiers, compound statements such as FOR...DO, IF...THEN, and ..WHILE.., the input and output statements, and the assignment statement. The only fig-FORTH command in this program is the (;S), which halts interpretation of the screen and, at run time, returns control to the calling procedure. It is not expected that you will understand the program just yet, but examining it will give a sense of Tiny-Pascal procedures. By the end of this series it will all be made clear.

If you feel adventurous, you may enter the program on three consecutive screens (each screen is blocked out in the listing) on the Tiny-Pascal starter screen set, using the editor. Check each screen for completeness and correct punctuation. FLUSH all changes to disk. To compile these screens, type "n LOAD", where n is the first screen used in the set. Interpret any error messages and make the necessary corrections. After a successful compilation, type the program name, IHAND, to run the program. To list the program to a printer, first set the Baud rate (see above for instructions), then use the "n 3PRINT" command, where n is the first screen number. To send the program output to the printer, type "1 PRINTF !", followed by the program name, "IHAND". The 3PRINT command was used to print the listing shown here.

I hope you will obtain the resources required for this series, and join with us as we explore this valuable language. Until next time you can practice the fig-FORTH and line editor commands, presented in this article, using a blank screen. With this experience you'll be ready for the next article. If you have any questions, feel free to call me between 9:00 PM and 11:00 PM (EST), Monday-Friday, at 716-889-4994. □

References:

1) "Installation Documentation for FORTH in the Compucolor II.", version 1.1.0. James C. Minor, 1982

2) "Installation Documentation for Tiny-Pascal in fig-FORTH for the Compucolor II," James C. Minor, 1983

3) "FORTH FUNDAMENTALS," Volume 1. C. Kevin McCabe, Dilithium Press, 1983.

4) "Introduction to Pascal and Structured Design." N. Dale and D. Orshalick. D.C. Heath and Company, 1983.

News from CUWEST:

The West Australia Compucolor/Intecolor User Group Newsletter for Jan/Feb/Mar of 1984 shares our own concern about the future of the CCII. After a review of pros and cons, their feeling is that the CCII is still a viable machine, because of its comparative versatility and cost, when viewing the currently available alternatives. The facilities available in Australia are much less those those we enjoy in the USA. Australian users have begun sharing software, including commercial issues, which are now being placed in user group libraries.

Australian computer owners are about to gain access to a national VIDEOTEX network. An attempt is being made to program the CCII to act as an access terminal. The graphics provisions in VIDEOTEX are sophisticated compared with The Source and Compuserve, so much so, in fact, that the CCII will be a compromise, at best, because of its limited pixel capacity. Online banking, shopping, ticketing, news, weather and sports will be offered in time. The baud rate of 1200 will make modems more costly to buy for the Australian user.

CUWEST has a current mailing list of about 40 people, including some in the USA and Canada. The last 8-page quarterly reprinted Gary Dinsmore's article from the last COLORCUE, on Basic's variables, and a "Tech Tip" from Ken Winder. Ken is recommending the replacement of Q6 and Q7 on the analog board with high beta devices. Their effect is to greatly reduce the CRT display compression during disk access, and to reinforce a good hardware reset at "turn-on" time. No modifications are required other than the simple exchange of these two parts. (No mention is made of the exact type to be used, but COLORCUE will look into this and report on it soon.)

```
SCR # 10
  0 (  IHAND  ) PASCAL
  1 PROGRAM IHAND;
  2 CONST
  3    CMP=10;
  4 VAR
  5    TEMP:INTEGER;
  6    I,K,L,M:INTEGER;
  7    J,N,P,I1,J1,A1,A2:INTEGER;
  8    IARAY:ARRAYE  10 ] OF INTEGER;
  9 BEGIN NEWLINE; NEWLINE; WRITE( 'ENTER ',#CMP,' INTEGERS:'  );
 10    FOR I:=1 TO CMP DO
 11      BEGIN NEWLINE;
 12        WRITE('ENTER NO. ',#I); READ( #N ); IARAYE I ]:=N
 13      END;
 14    I1:=CMP;
 15    WHILE I1>1 DO
```

```
SCR # 11
  0      BEGIN
  1        J1:=1;
  2        WHILE J1<I1 DO
  3          BEGIN
  4            A1:=IARAYE J1 ]; A2:=IARAYE J1+1 ];
  5            IF A1>A2
  6              THEN
  7                BEGIN
  8                  TEMP:=IARAYE J1 ];
  9                  IARAYE J1 ]:=IARAYE J1+1 ];
 10                  IARAYE J1+1 ]:=TEMP
 11                END;
 12            J1:=J1+1
 13          END;
 14        I1:=I1-1
 15      END;
```

```
SCR # 12
  0    J:=1; NEWLINE;
  1    WHILE J<10 DO
  2      BEGIN NEWLINE;
  3        K:=J+1; L:=IARAYE J ]; M:=IARAYE K ]; P:= K DIV 2;
  4        WRITE( 'INTEGER PAIR # ',#P,' IS ',#L,#M );
  5        J:=J+2
  6      END;
  7 END. ;S
  8
  9
 10
 11
 12
 13
 14
 15
```

On Passing.....

[There is no escape from the plain fact that FOR SALE signs in Colorcue take on the aspect of an obituary. David Suits submitted his FOR SALE sign for this issue, but the equipment was purchased before press time. However, something needs to be said about this rather significant event in our lives, so I am printing the following letter exchange to mark the event. I believe I am representing the feelings of most of us, more or less accurately. ED]

Dear Editor,

Volume VI, Number 1 of COLORCUE arrived recently, and I was impressed beyond measure.

If you have back issues of COLORCUE—back to Volume 1, Number 1—it is instructive to look them over. Many of us who were around at the very beginning of things were ecstatic when COLORCUE was born. A simple, typed newsletter, a few pages long, it was put together by ISC. When it looked as thougn the Compucolor II was going to sell fairly well, COLORCUE took on a new, and a little more slick appearance.

When Ben Barlow and I took over COLORCUE more than two years ago, we were proud to inaugurate what we felt was an even better look to COLORCUE.

And now, when Compucolorists seem gradually to be drifting away from the GOM (Grand Old Machine), COLORCUE suddenly becomes more beautiful than ever before—a newsletter worthy of a circulation of thousands!

Congratulations, and best regards.

David Suits.

Dear David,

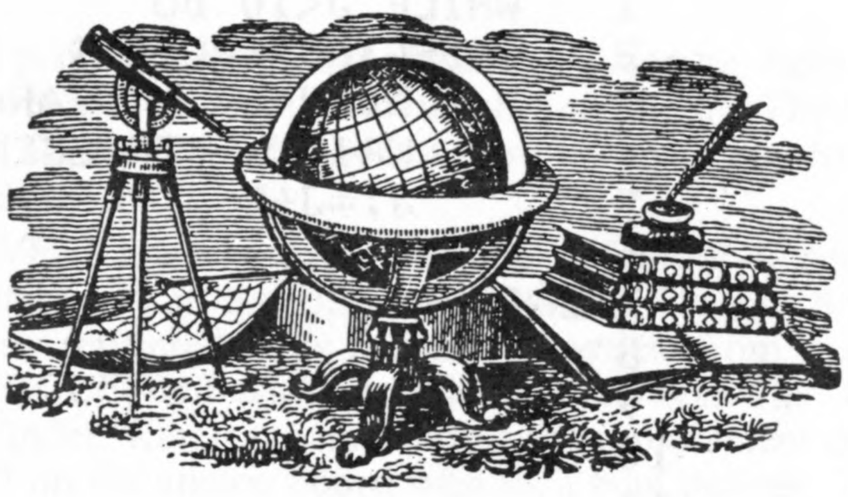
Thank you for your note of appreciation. Although we may sport a new format, made possible by a few fortunate circumstances, it is always the content that matters in any magazine. The two years under the editorship you and Ben gave us, at great cost of personal time and money, will be very challenging to reproduce. COLORCUE really 'became' a magazine under your auspices, and the quality of the articles was consistantly outstanding. It is a plain and simple truth that but for the two of you, we would all have been lost. That is a debt we owe you collectively.

While I'm sure many others will share my sadness at acknowledging your FOR SALE sign, you have surely 'earned' your passage on to other things, and we wish you very well indeed, and will continue to watch and think supportively of your computer career. As an inadequate parting gift, we are presenting you with some disks for your NEC (expensive, aren't they!' to be filled with even more spectacular graphics.

In the meantime, on we go! If we are to depart, we shall do it in style. The Compucolor community is blessed with an unusual collection of talent, as you know, and until the last CRT controller chip says farewell, we will make the most of what we have until we, too, must move on. Your legacy is going to carry us a good bit of the way. Thank you.

Joseph Norris

TRANSFER SWITCH: MFJ Enterprises Incorporated manufactures a line of transfer switches which permit switching your serial output to any one of several peripherals, such as a modem and printer. Various numbers of input and output combinations are available. One input to two outputs costs $79.95. Ten lines (wires) are switched for this price. Models are available that switch all 25 lines if required. The switch is wired to reverse transmit and receive lines (pins 2 and 3) on the RS232 bus. LEDs monitor the signal flow, and data surge protection is provided. A 30 day money-back guarantee is offered. Order from MFJ Enterprises, 921 Louisville Road, Starkville, MS 39759. (601) 323-5869. VISA, MASTERCARD, check or money order will do. Add $4.00 for shipping.



BACK COVER: The chart shown on the back cover was submitted by Ric Lowe. He uses it when examining assembly language trouble spots, to record the register contents at significant points. Ric says he also keeps paper handy for memory contents as well. We put it on the back so you can make ready copies from any copy machine.

UNCLASSIFIED ADVERTISEMENTS

FOR SALE: COMPUCOLOR II, v6.78. 32K memory, internal disk drive, standard keyboard with F0-F15 keys added (but no numeric or color keypads). Other items included are Maintenance Manual, all back issues of Colorcue, Personal Data Base, Basic Language 1-10, Formatter, Basic Editor, Screen Editor, Assembler, Fortran, and miscellaneous games and utilities. The computer is in good working order. Asking $700 (shipping prepaid) by money order or certified check. Bill Anthony, 655 East Wells Way, Camano Island, WA 98292. (206) 387-1576.

FOR SALE: Compucolor II, v6.78, 32K, extended keyboard, manuals including Programming, Maintenance, "Color Graphics" and "Basic Training." 15 disks included. Excellant condition. Asking $1000. Art Tack. 1127 Kaiser Road, SW, Olympia, WA 98502.

The following items are from the estate of Myron T. Steffy.

FOR SALE: Compucolor II computer system, v6.78, with full keyboard, 32K memory, two disk drives, Devlin analog protector, 2 Devlin RAM cards with software switch, lower case character set and character generator for FREDI. Excellant condition. $1200.

Morrow MicroDecision CP/M computer with dual disk drives, 64K memory and software package. No monitor. $900.

Novation CAT, 300 Baud modem, Votrax unit, TI SR52 calculator and TI programmer's calculator, 2 MFT transfer switches for RS232 outputs. All in good condition. Make offer.

Diablo Model 630 letter quality printer. Cost $1850 wher new. Excellant condition. Make offer.

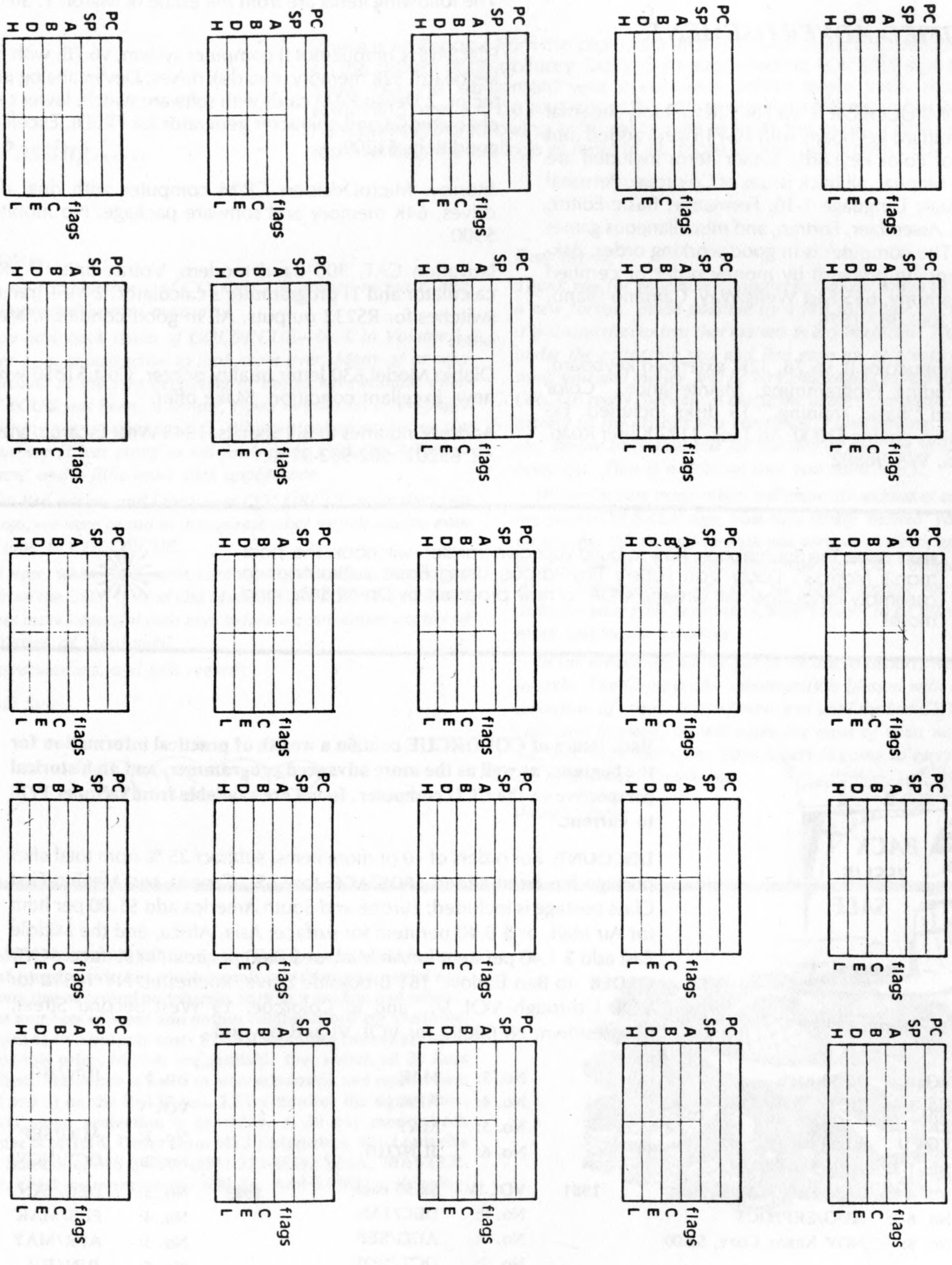
Address inquiries to Bill Shanks. 1345 West Escarpa, Mesa AZ 85201. 602-962-0130.

NEXT ISSUE: The Sourcebook; How to build your own transfer switchbox; The 'poor man's Morrow'; Doug Van Putte's Tiny-Pascal; Using Basic subroutines in assembly language; Bill Greene's IDA; a new program by David Suits; and more!

Back issues of COLORCUE contain a wealth of practical information for the beginner as well as the more advanced programmer, and an historical perspective on the CCII computer. Issues are available from October 1978 to current.

DISCOUNT: For orders of 10 or more items, subtract 25 % from total after postage has been added. **POSTAGE:** for U.S., Canada and Mexico First Class postage is included; Europe and South America add $1.00 per item for Air Mail, or $ 0.40 per item for surface; Asia, Africa, and the Middle East add $ 1.40 per item for Air Mail, or $ 0.60 per item for surface. **SEND ORDER** to Ben Barlow, 161 Brookside Drive, Rochester. NY 14618 for VOL I through VOL V; and to Colorcue, 19 West Second Street, Moorestown, NJ 08057 for VOL VI and beyond.

1978	**VOL I**	**$3.50 each**
	No. 1-3:	OCT/ NOV/ DEC
1979	**VOL II**	**$3.50 each**
	No. 1-3:	APR/MAY/JUN
	No. 4-5:	JAN/FEB/MAR
	No. 6-7:	AUG/SEP/OCT
	No. 8:	NOV XEROX COPY, $2.00
1980	**VOL III**	**$1.50 each**
	No. 1	DEC/JAN
	No. 2:	FEB
	No. 3:	MAR
	No. 4:	APR
	No. 5:	MAY
	No. 6:	JUN/JUL
1981	**VOL IV**	**$2.50 each**
	No. 0:	DEC/JAN
	No. 1:	AUG/SEP
	No. 2:	OCT/NOV
1982	No. 3:	DEC/JAN
	No. 4:	FEB/MAR
	No. 5:	APR/MAY
	No. 6:	JUN/JUL
	VOL V	
	No. 1:	AUG/SEP
	No. 2:	OCT/NOV
1983	No. 3:	DEC/JAN
	No. 4:	FEB/MAR
	No. 5:	APR/MAY
	No. 6:	JUN/JUL
1984	**VOL VI**	**$3.50 each**


PC
SP
A
B
D
H
flags
C
E
L